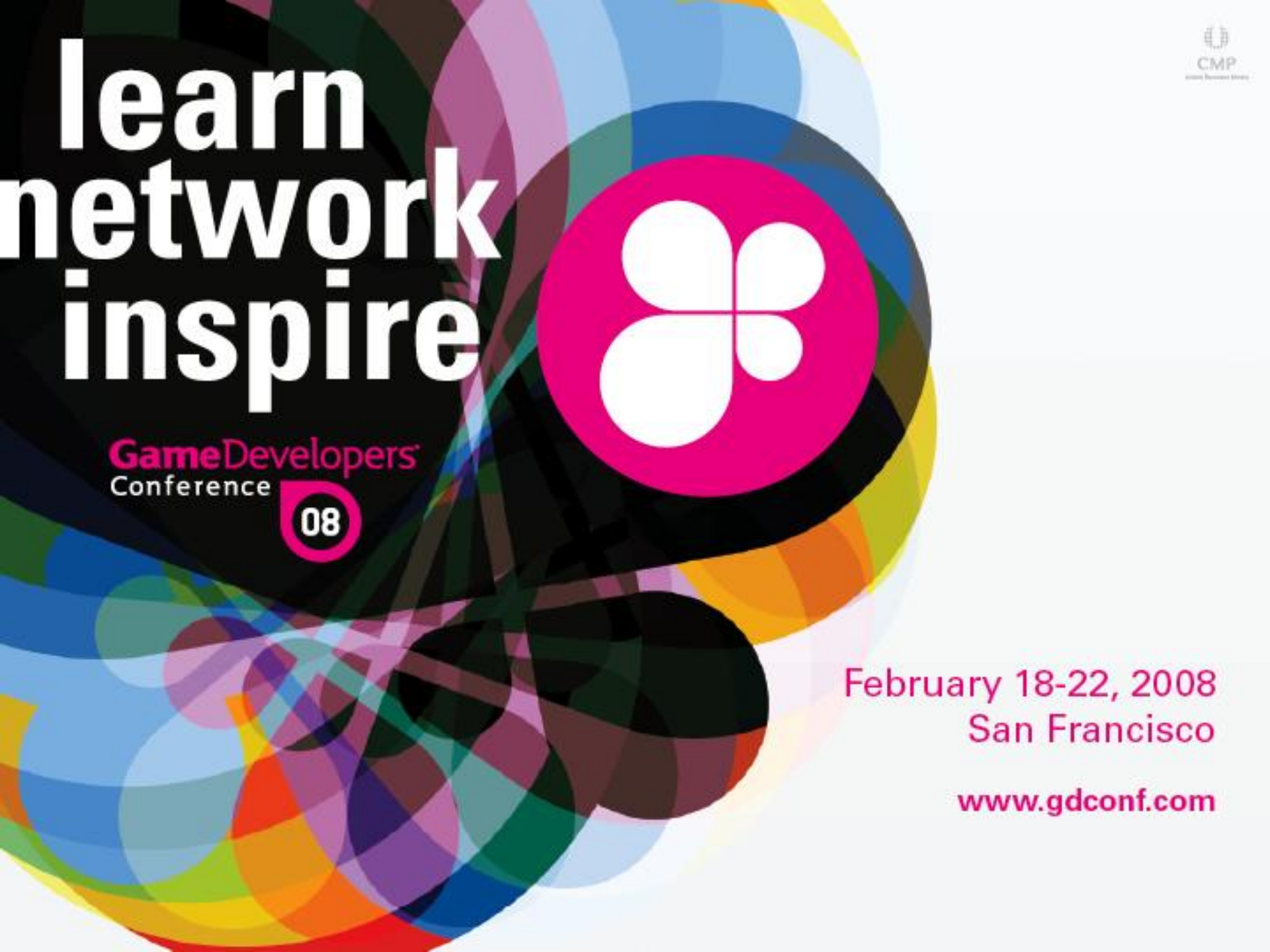

learn
network
inspire
GameDevelopers
Conference
08
CMP
February 18-22, 2008
San Francisco
www.gdconf.com

Harnessing the Power of Multiple GPUs

Games Developer Conference 2008
Holger Gruen, holger.gruen@amd.com

Material:
Holger Gruen,
Jon Story, jon.story@amd.com
Ignacio Llamas, illamas@nvidia.com

Agenda

- Why MGPU?
- Driver/Programming Considerations
- Common Pitfalls & Solutions
- Conclusions
- Q&A

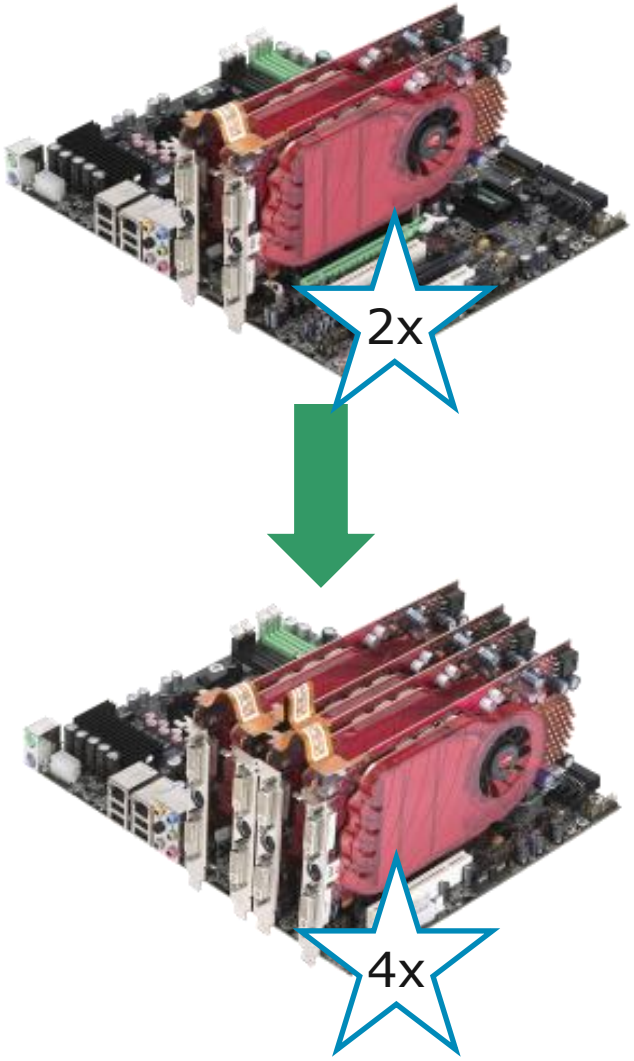
Why MGPU?

Why MGPU?

- MGPUs can be used to dramatically increase performance and visual quality
  - At higher screen resolutions
  - Especially with increased use of MSAA
- Many applications become GPU limited at higher screen resolutions/ quality settings
  - High resolution monitors => mainstream affordability
- Achieve next generation performance on today's HW
  - Prototype your next engine
  - Speed up your development time

MGPU configurations

- Multiple Boards
- Multiple GPUs per Board
- Hybrid MGPU

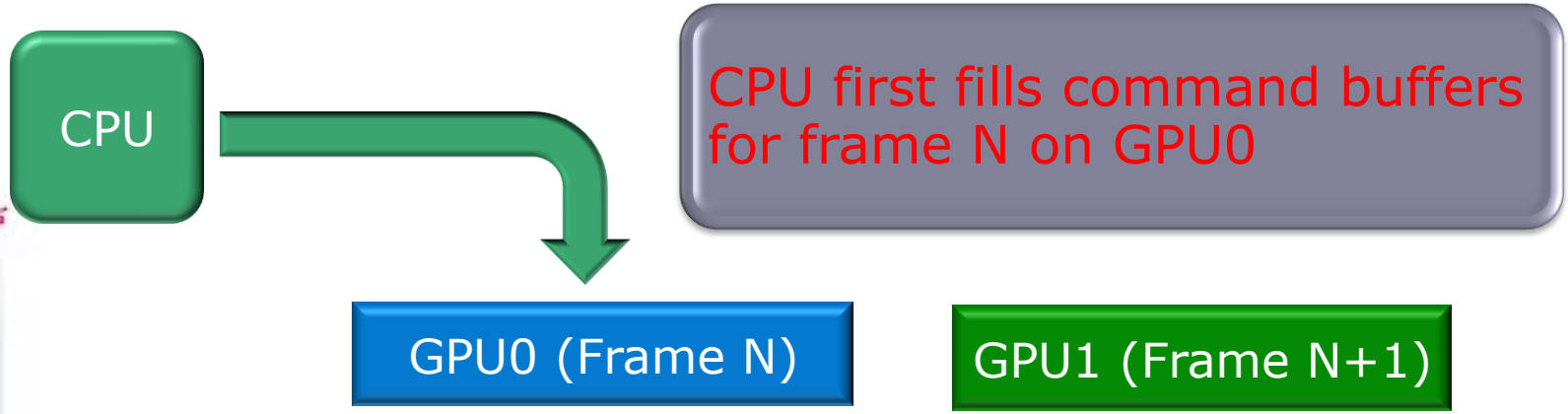


MGPU Rendering Modes

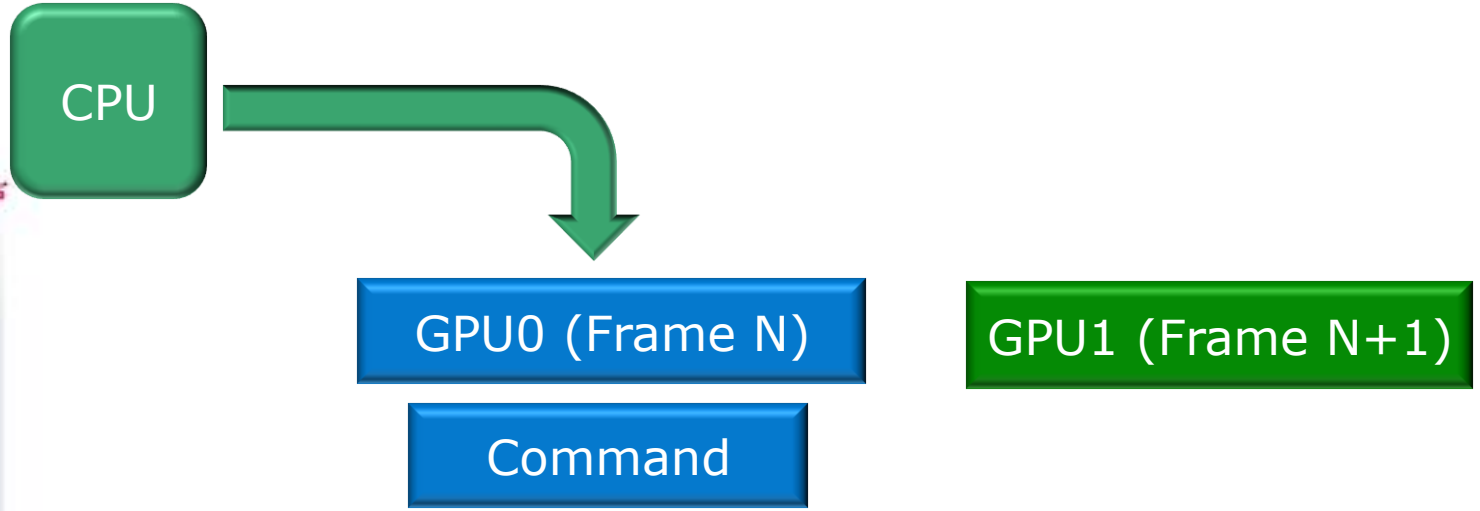
- Split Frame Rendering / Scissor
  - Screen is divided between GPUs
  - Dynamic load balancing

- Alternate Frame Rendering
  - GPUs take alternate frames
  - Highest performing mode

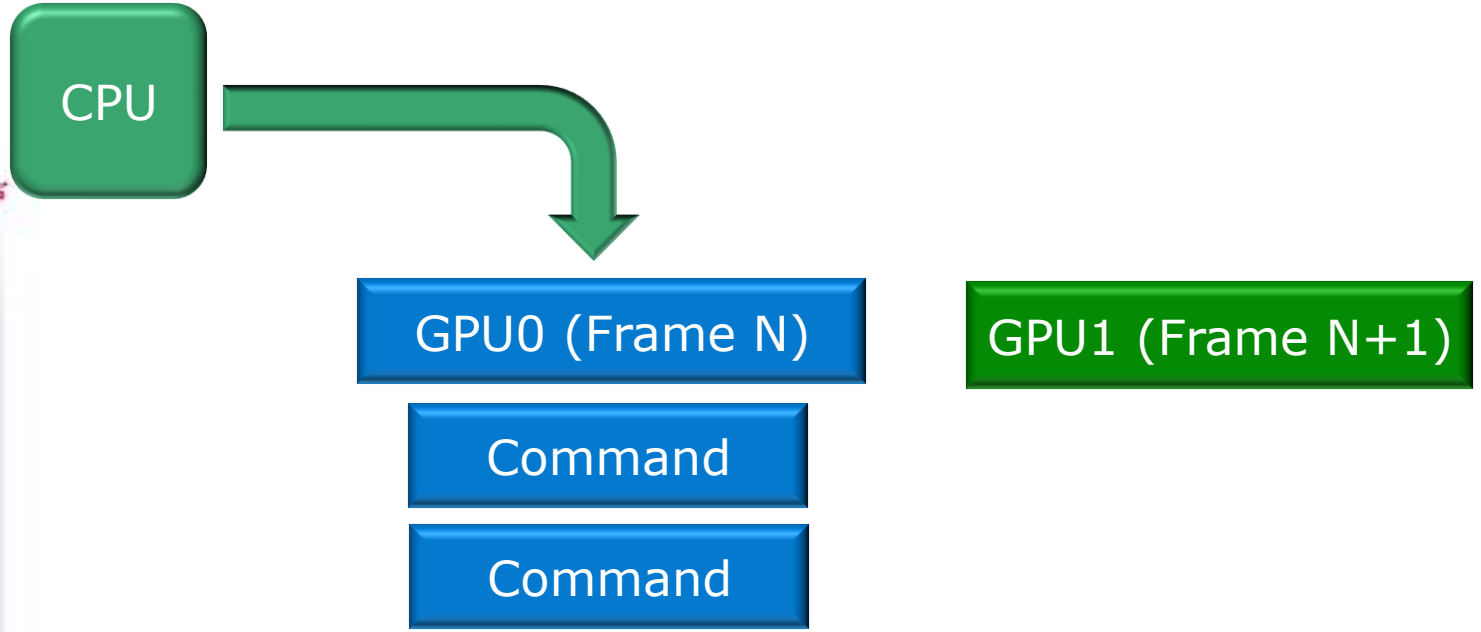
How does AFR Work?



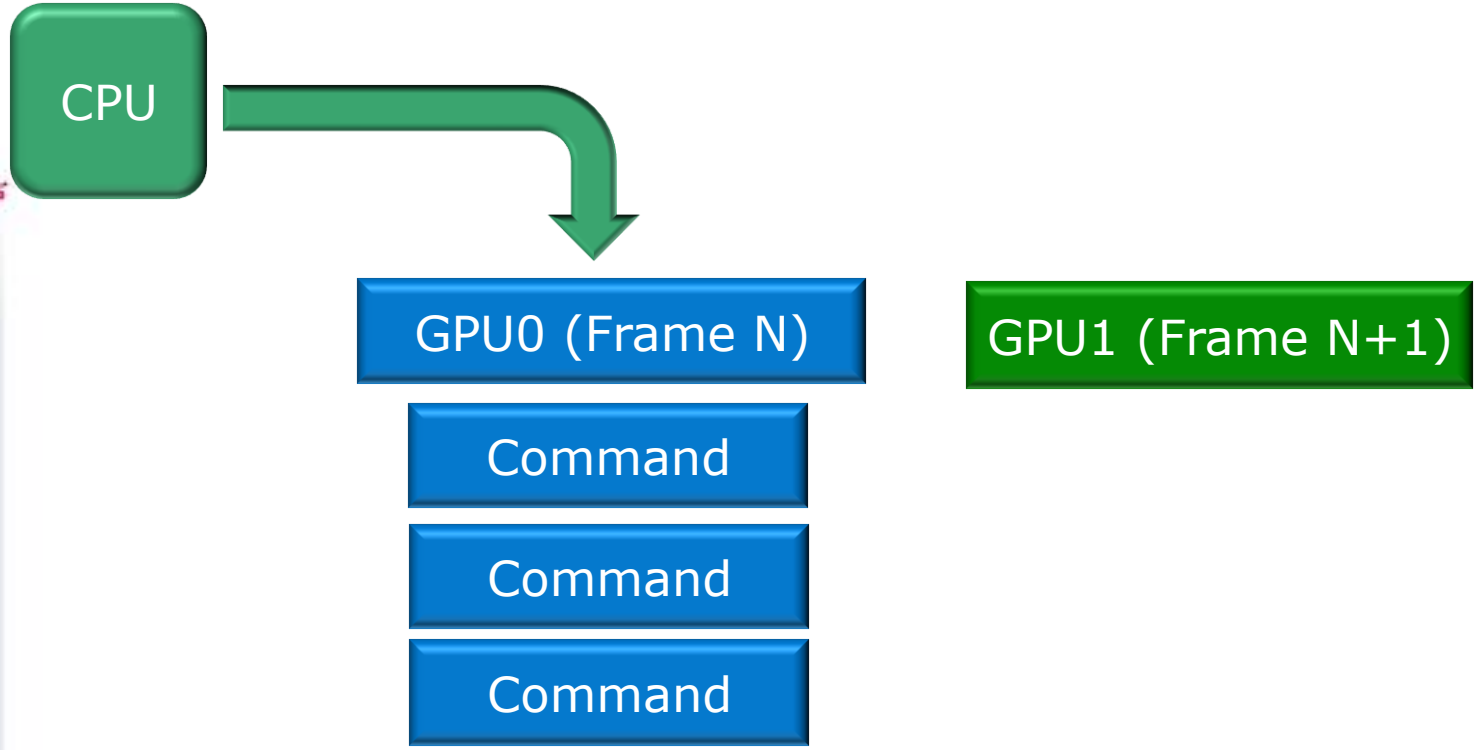
How does AFR Work?



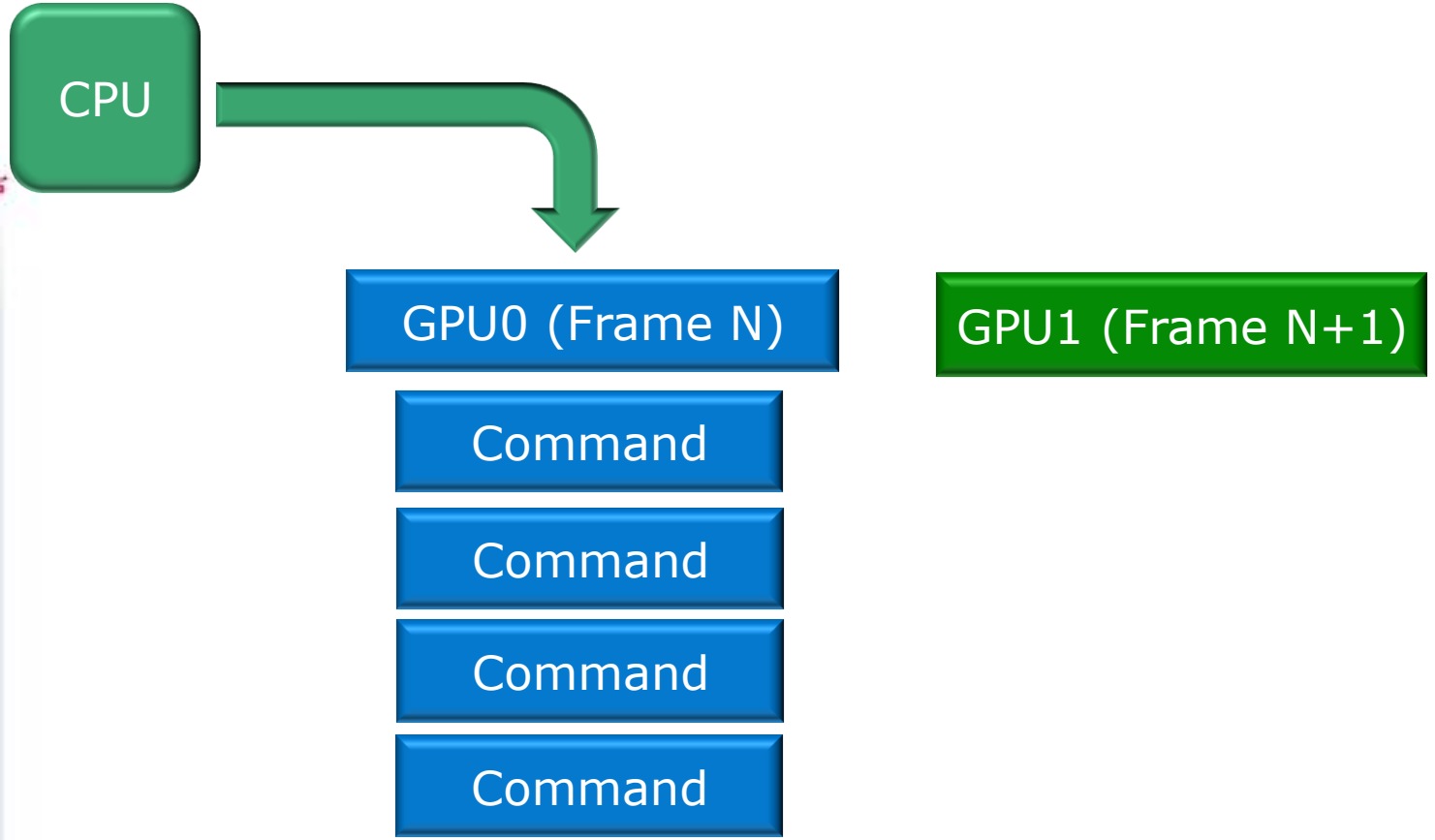
How does AFR Work?



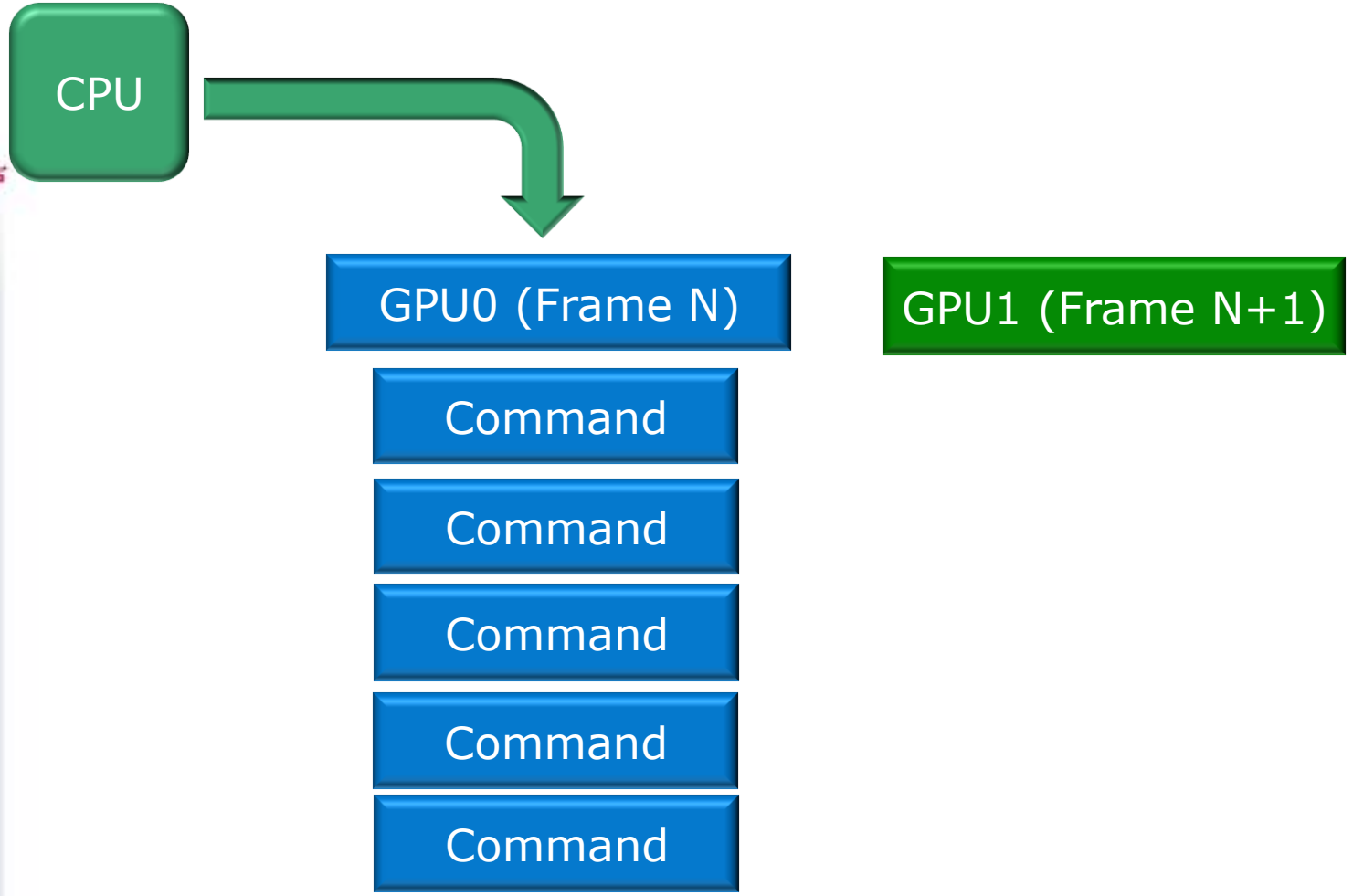
How does AFR Work?



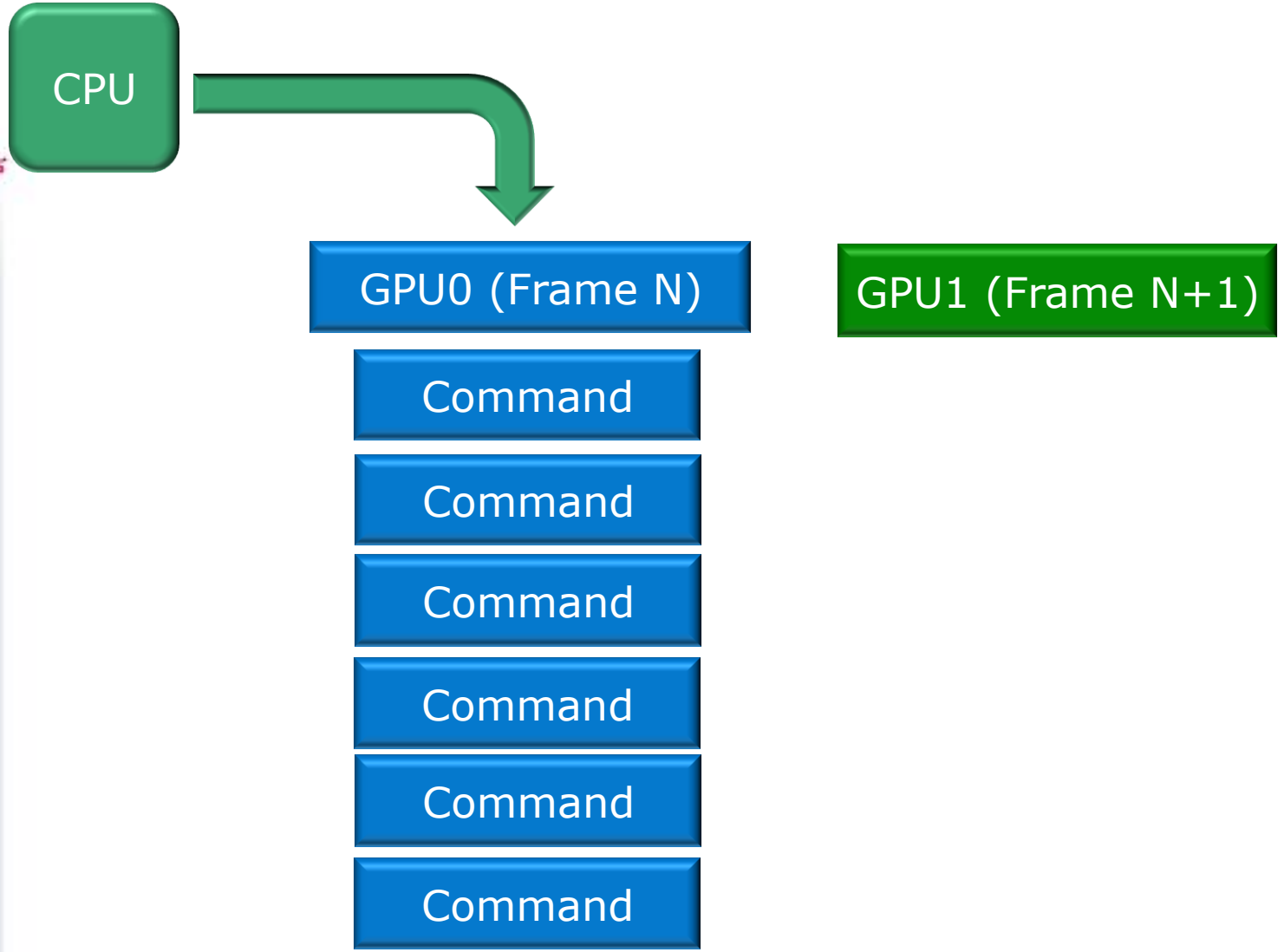
How does AFR Work?



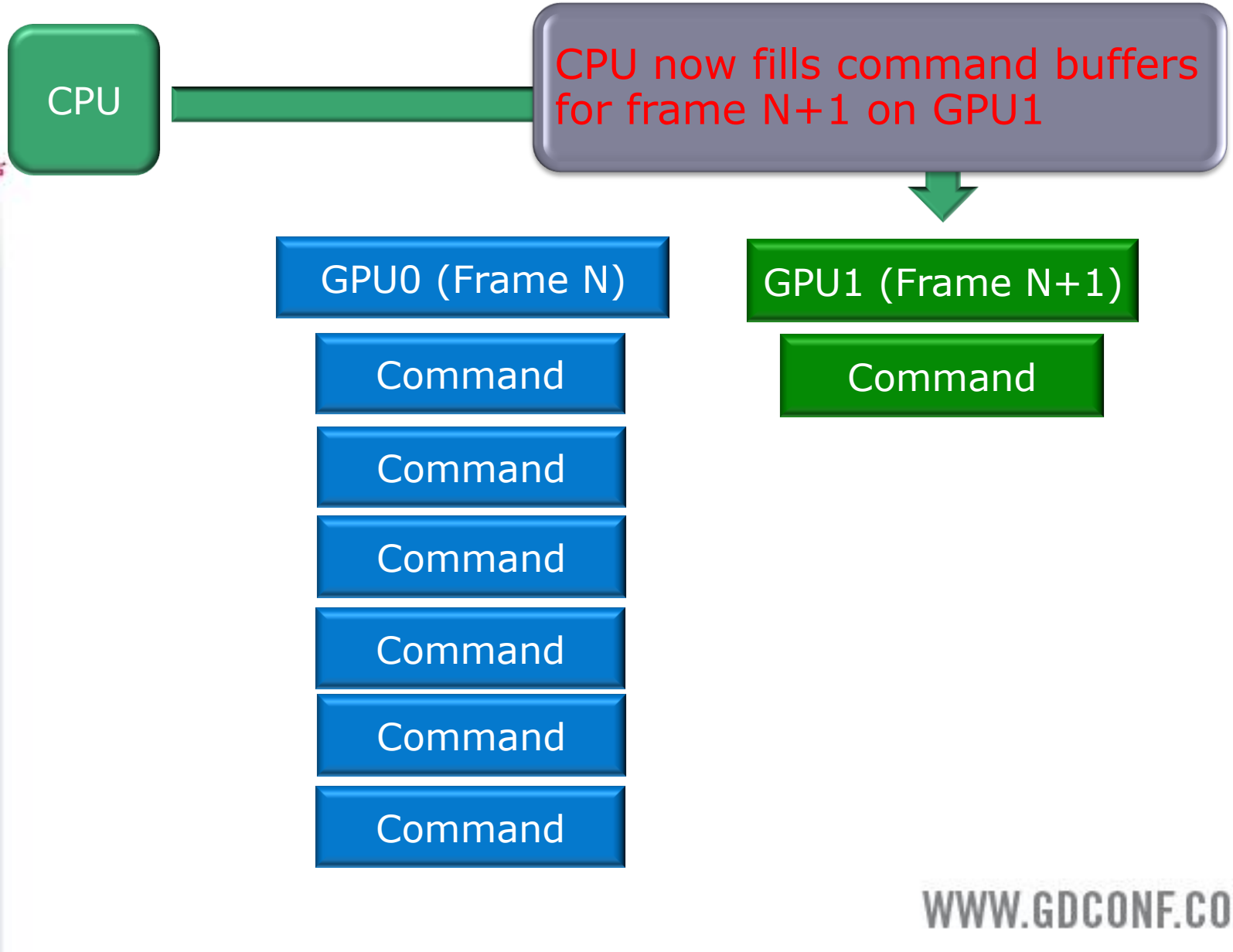
How does AFR Work?



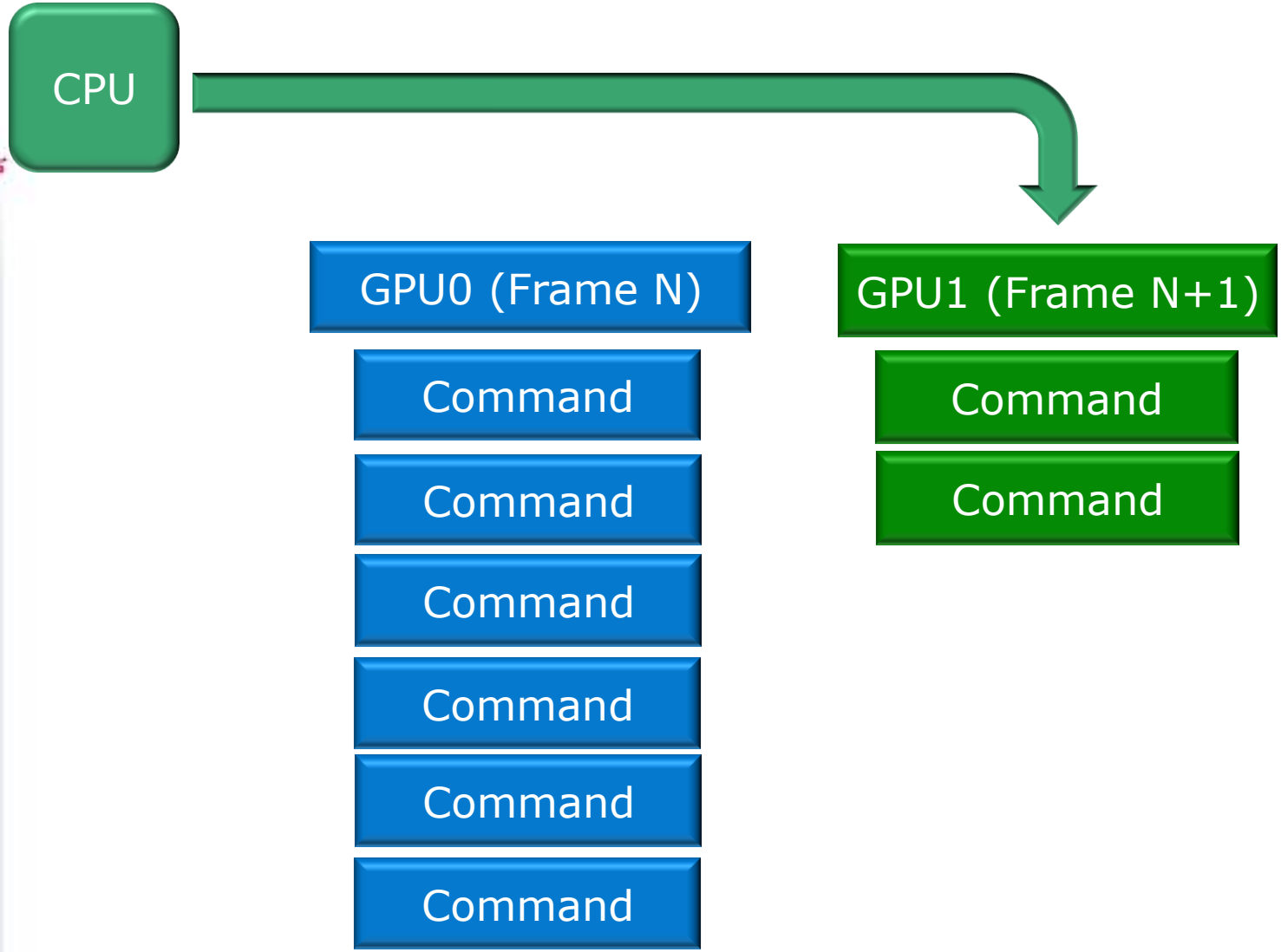
How does AFR Work?



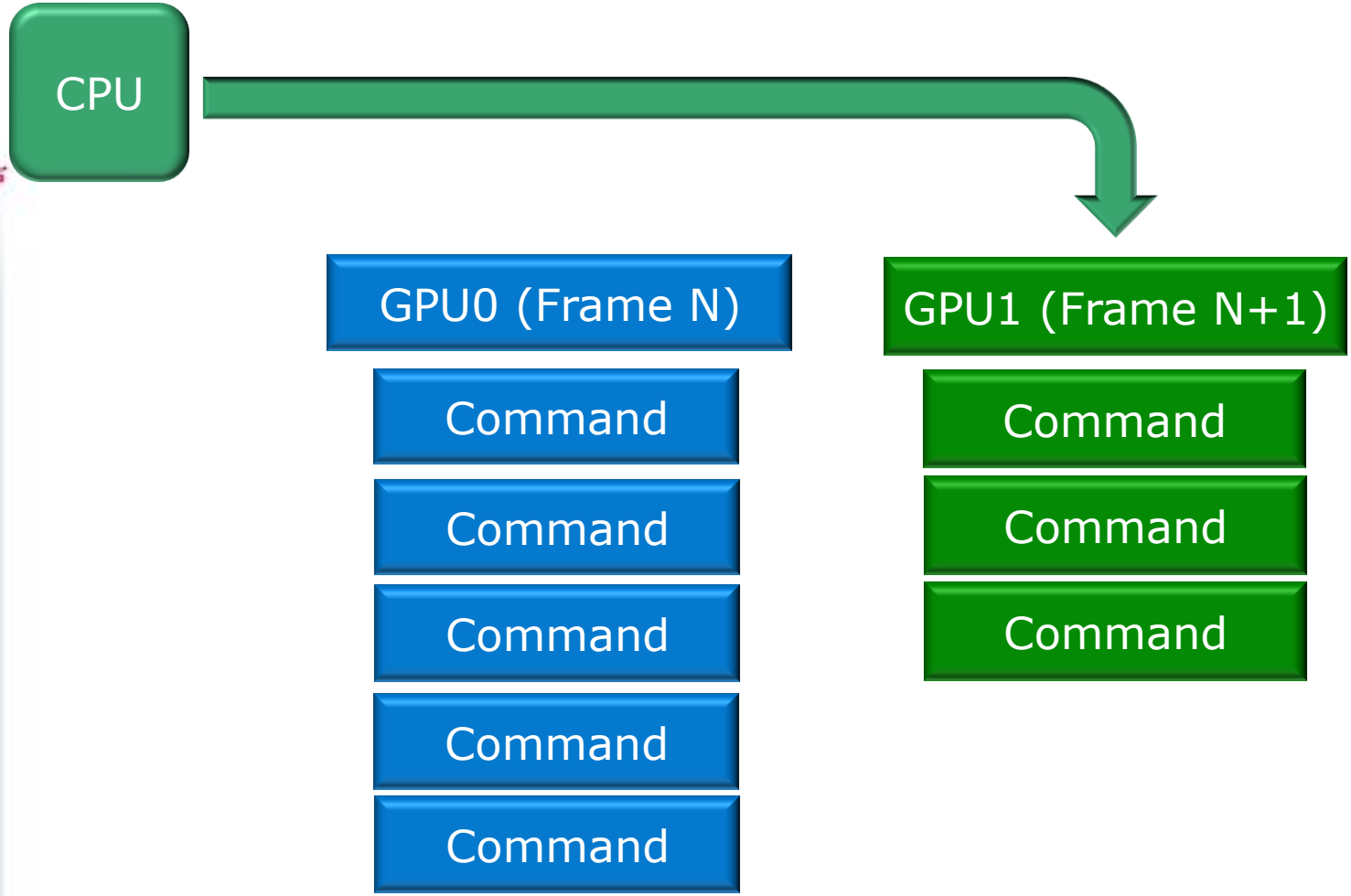
How does AFR Work?



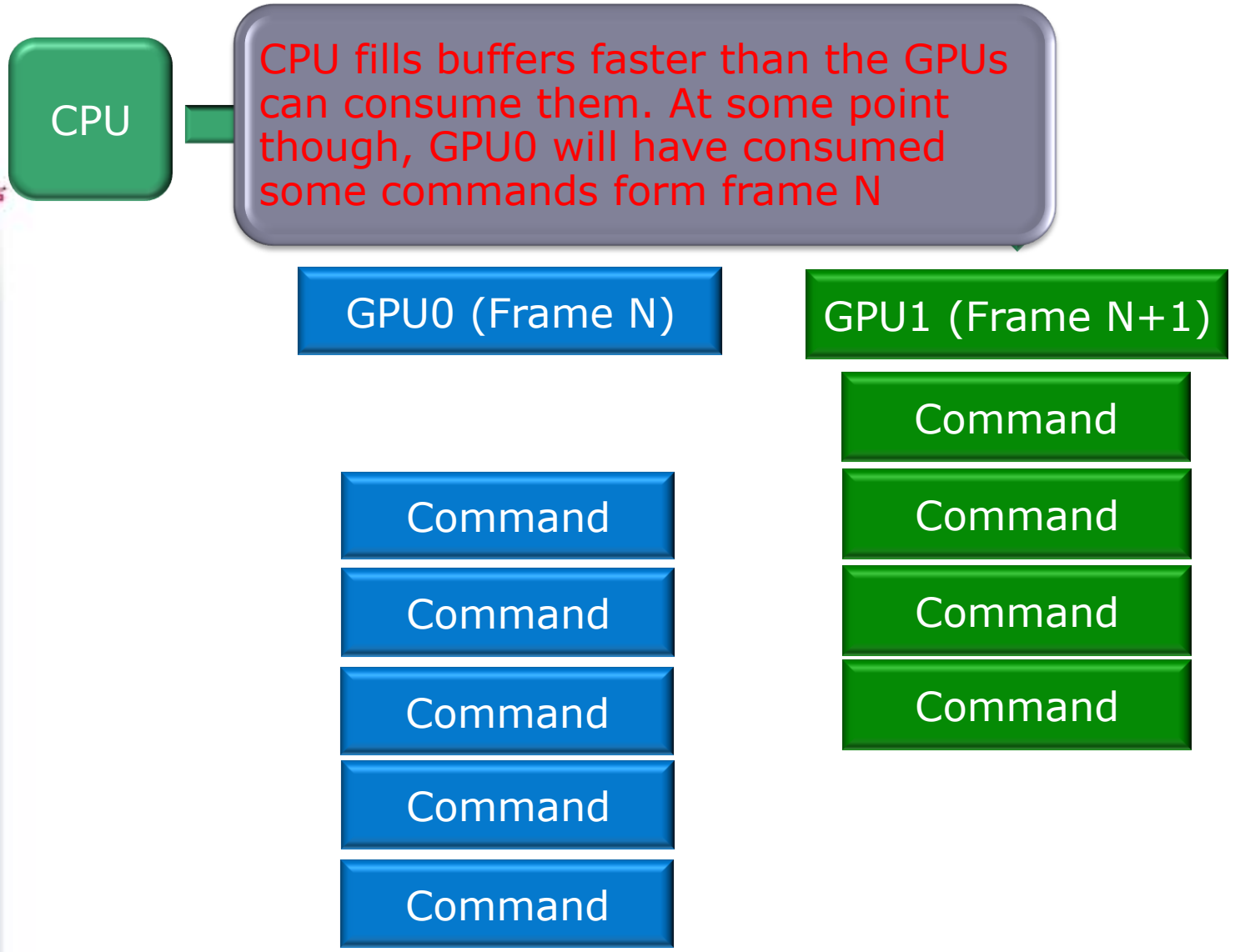
How does AFR Work?



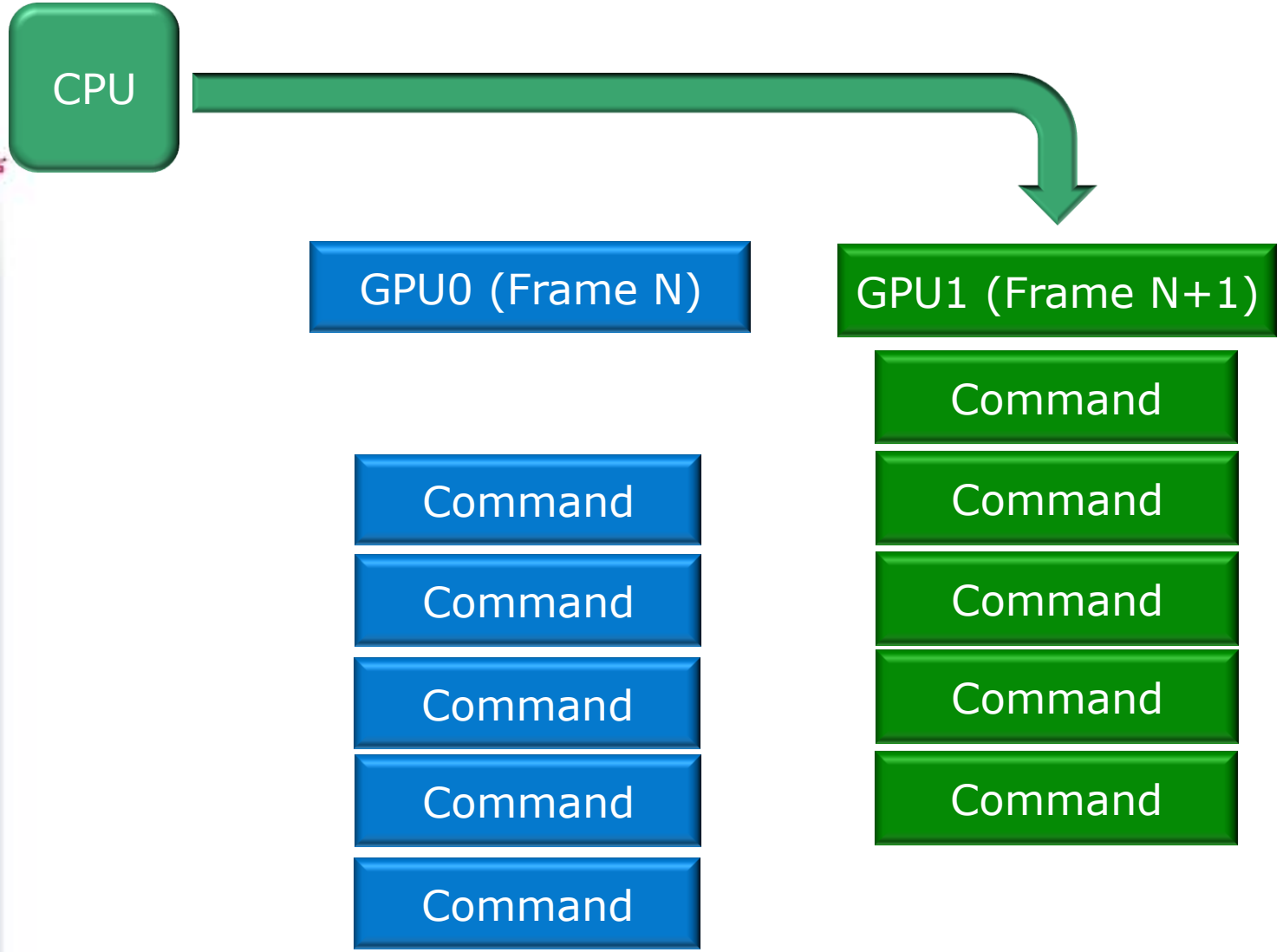
How does AFR Work?



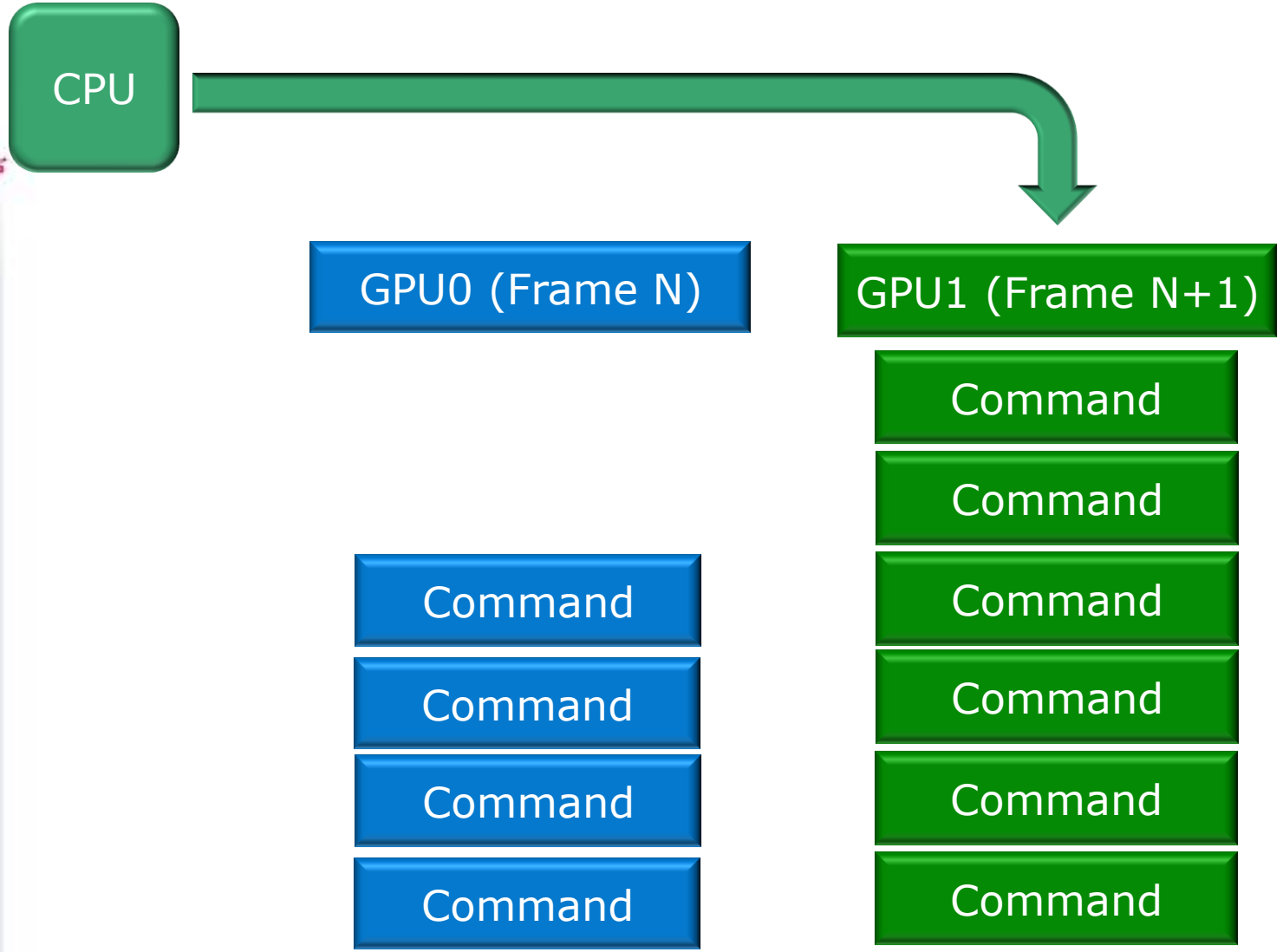
How does AFR Work?



How does AFR Work?



How does AFR Work?



Driver/Programming Considerations

Driver modes for MGPU

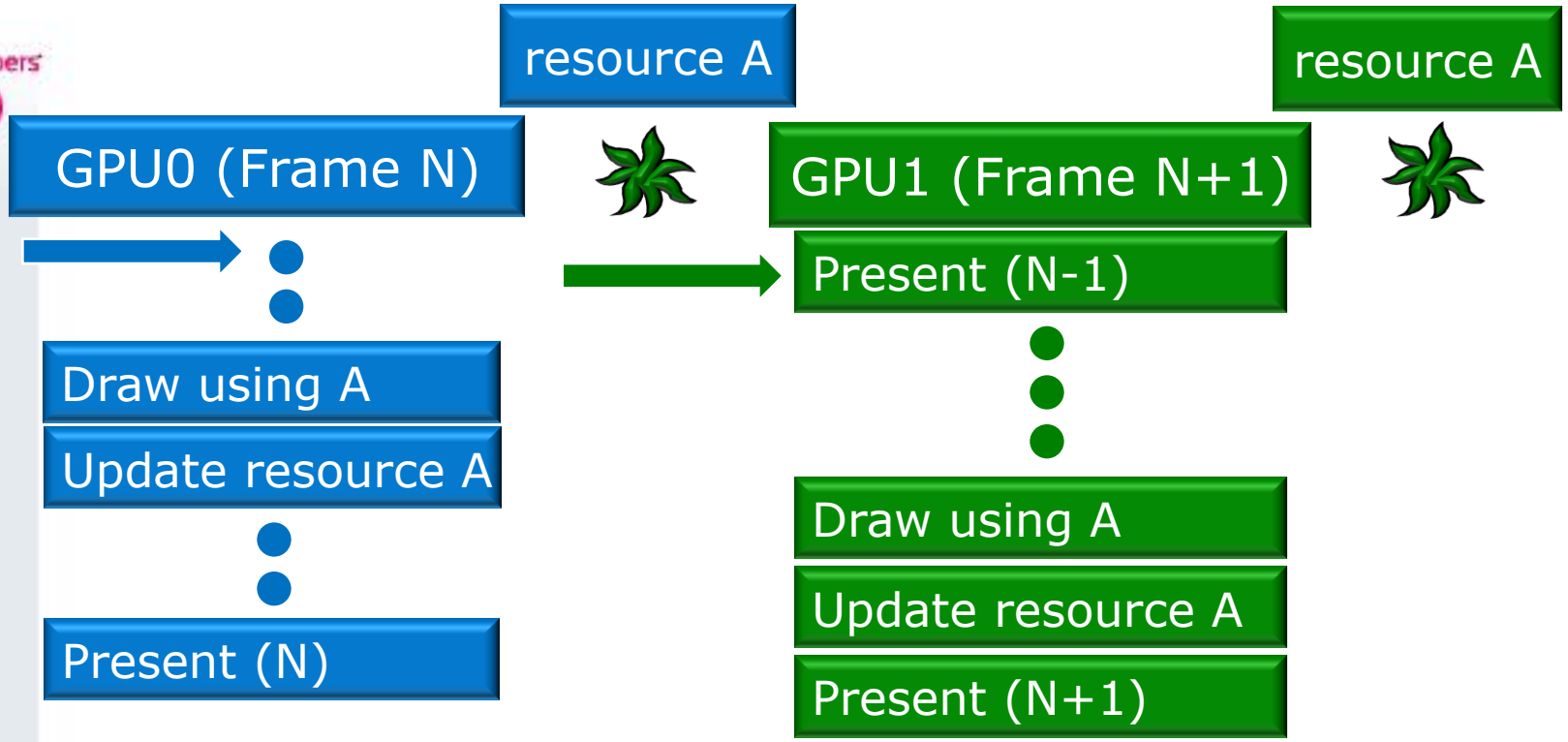
- Compatible AFR Mode
  - Default mode
  - Driver checks for AFR unfriendly behaviour
  - Will work around AFR unfriendly behaviour
- Full AFR Mode (Application Profile)
  - Driver recognises EXE name
  - Behaviour fully guided by profile
  - Best performance – no checking
  - We need developer input here!
  - Rename EXE to "AFR-FriendlyD3D.exe"
    - No checking : Speed & compatibility test

Programming for MGPU

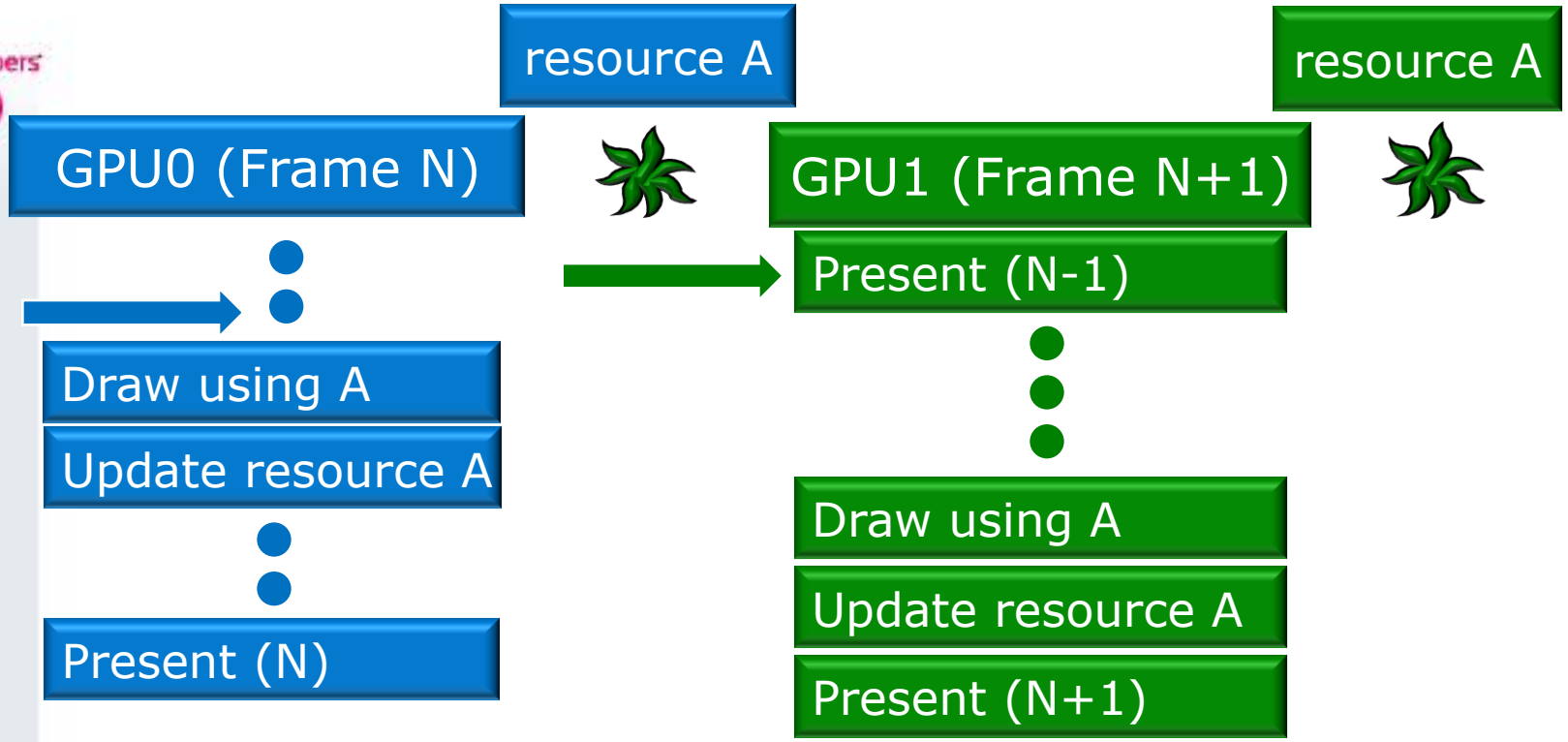
- Current MGPU setups are not shared memory architectures
  - Apps need to behave well to scale
  - Need to keep resources in sync on all GPUs
- Know what GPU(s) are rendering the current frame
  - Critical to adapt application behavior
- Use AMD / NVIDIA libraries
  - AMD: Query the number of GPUs
  - Nvidia: Query MGPU topology

Common Pitfalls & Solutions

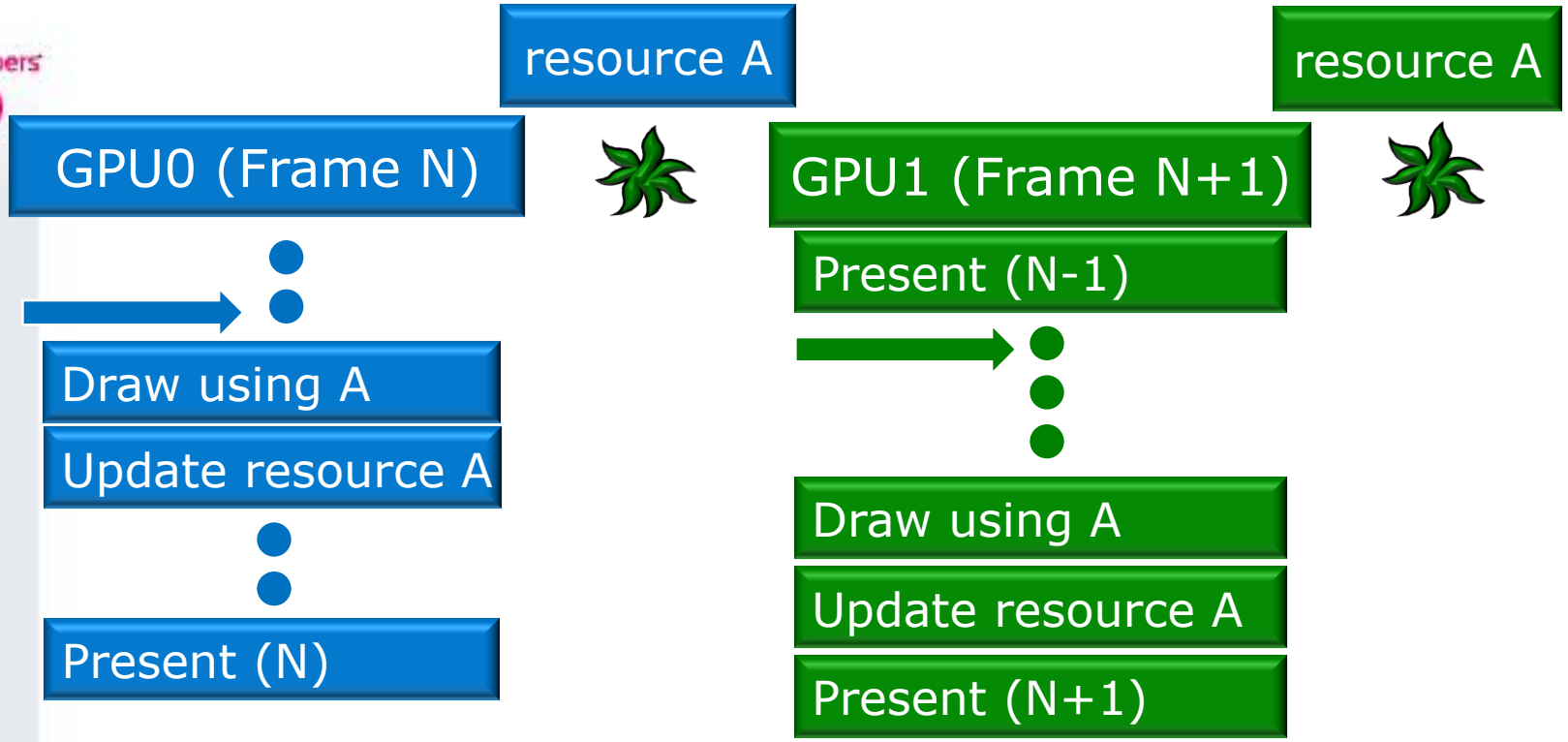
Pitfall: Dependencies Between Frames



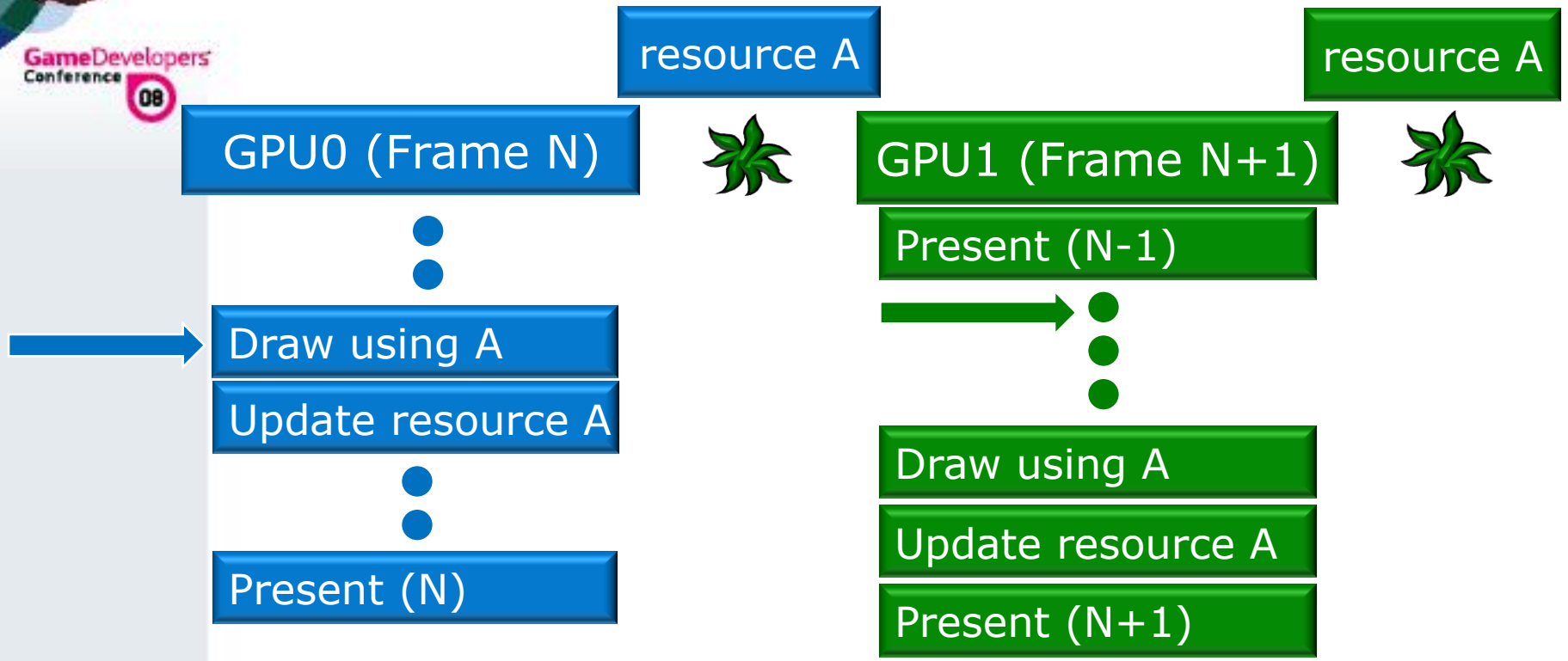
Pitfall: Dependencies Between Frames



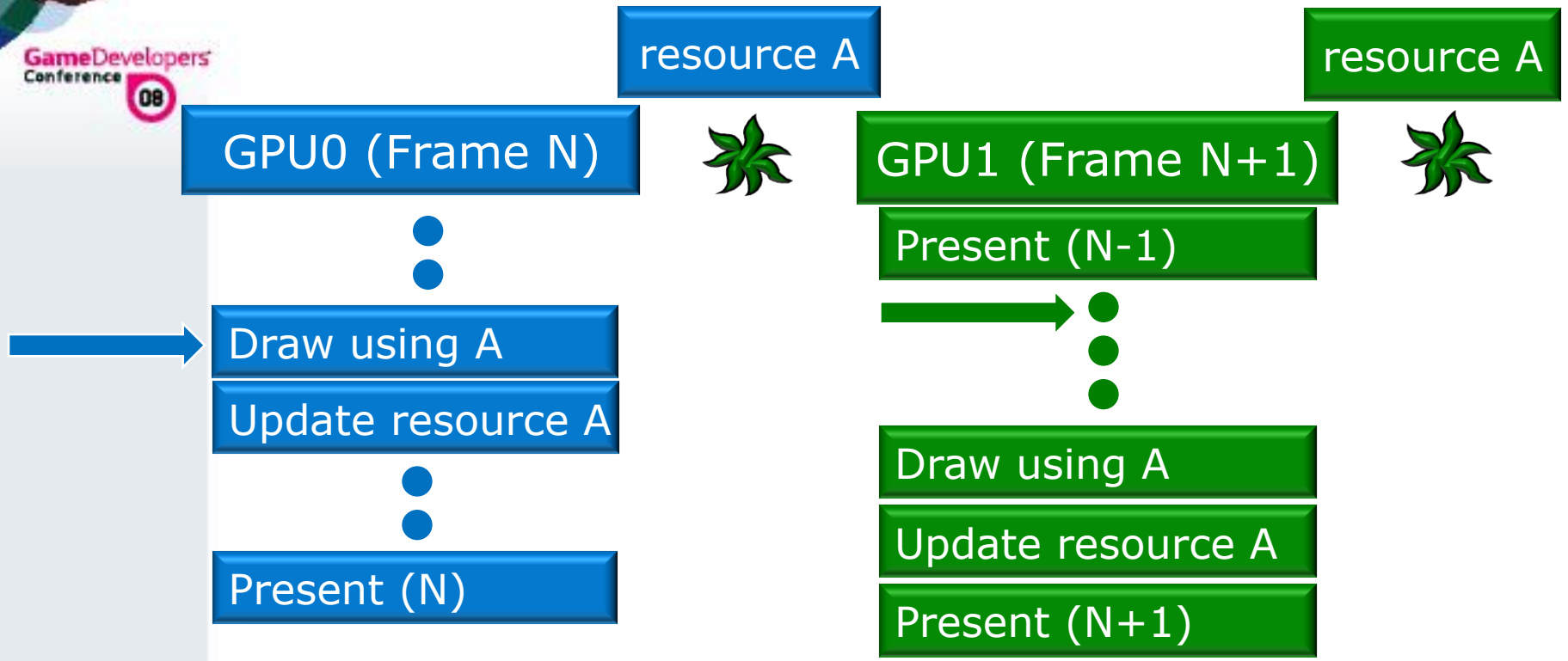
Pitfall: Dependencies Between Frames



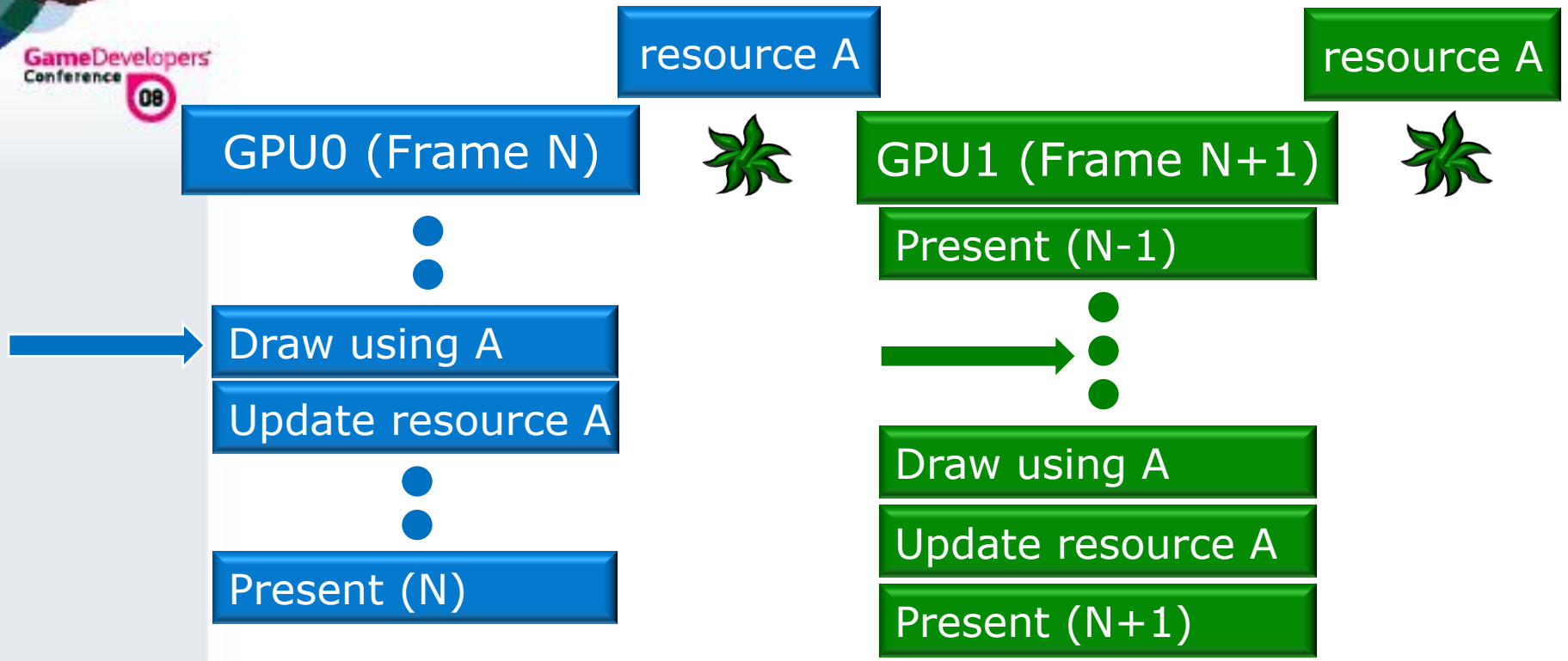
Pitfall: Dependencies Between Frames



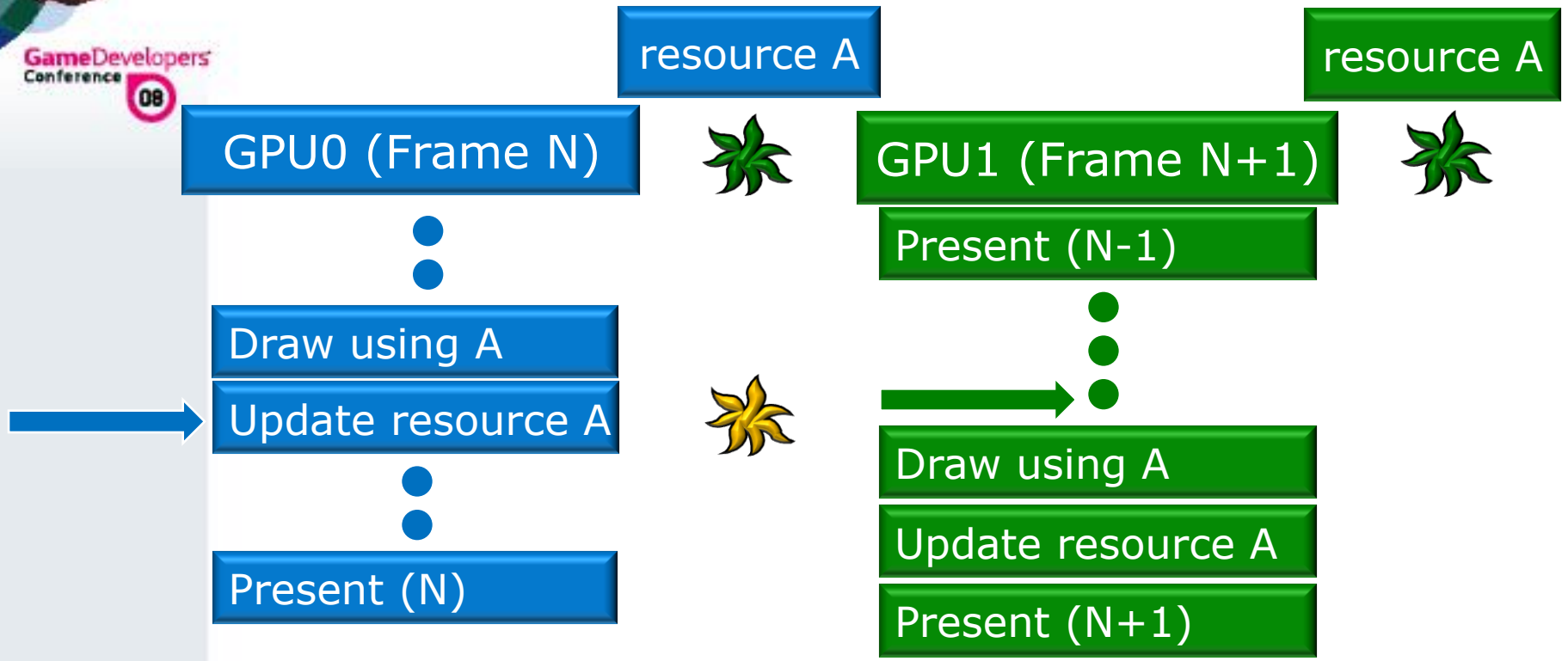
Pitfall: Dependencies Between Frames



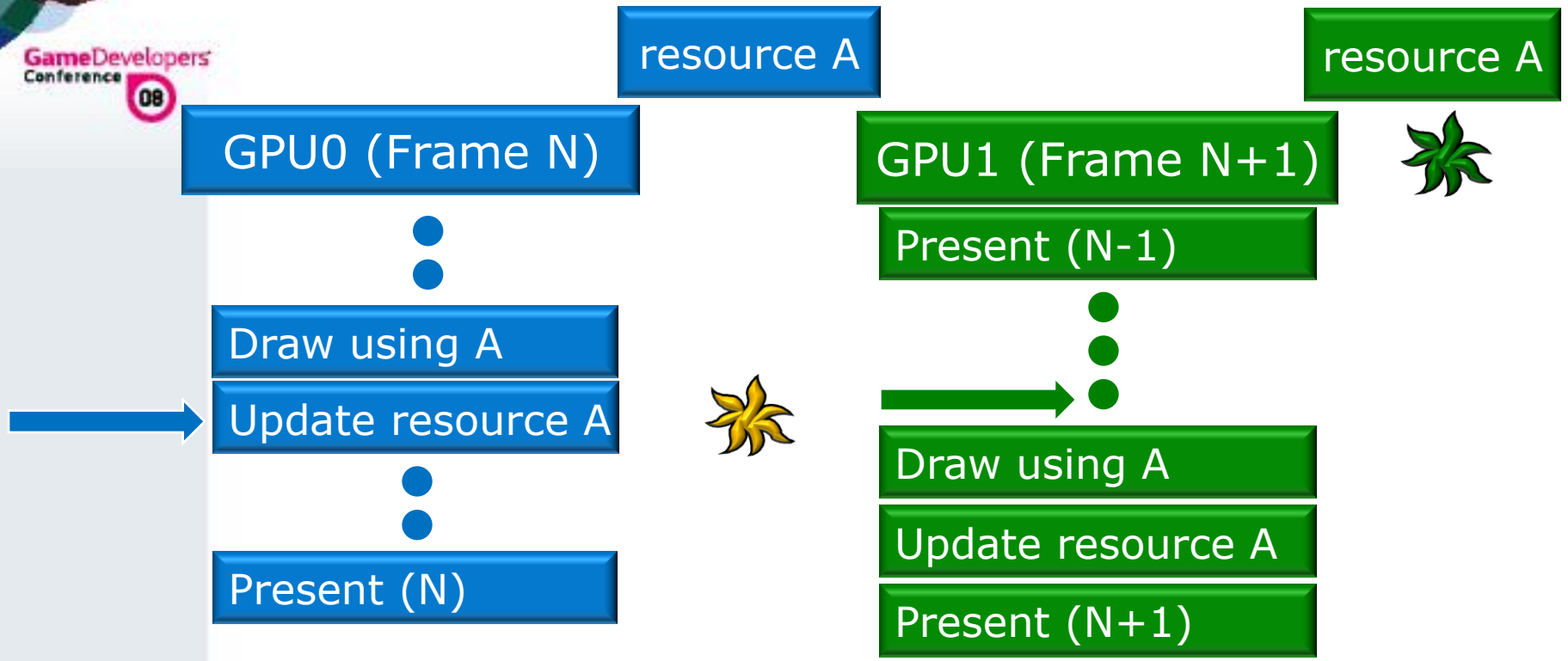
Pitfall: Dependencies Between Frames



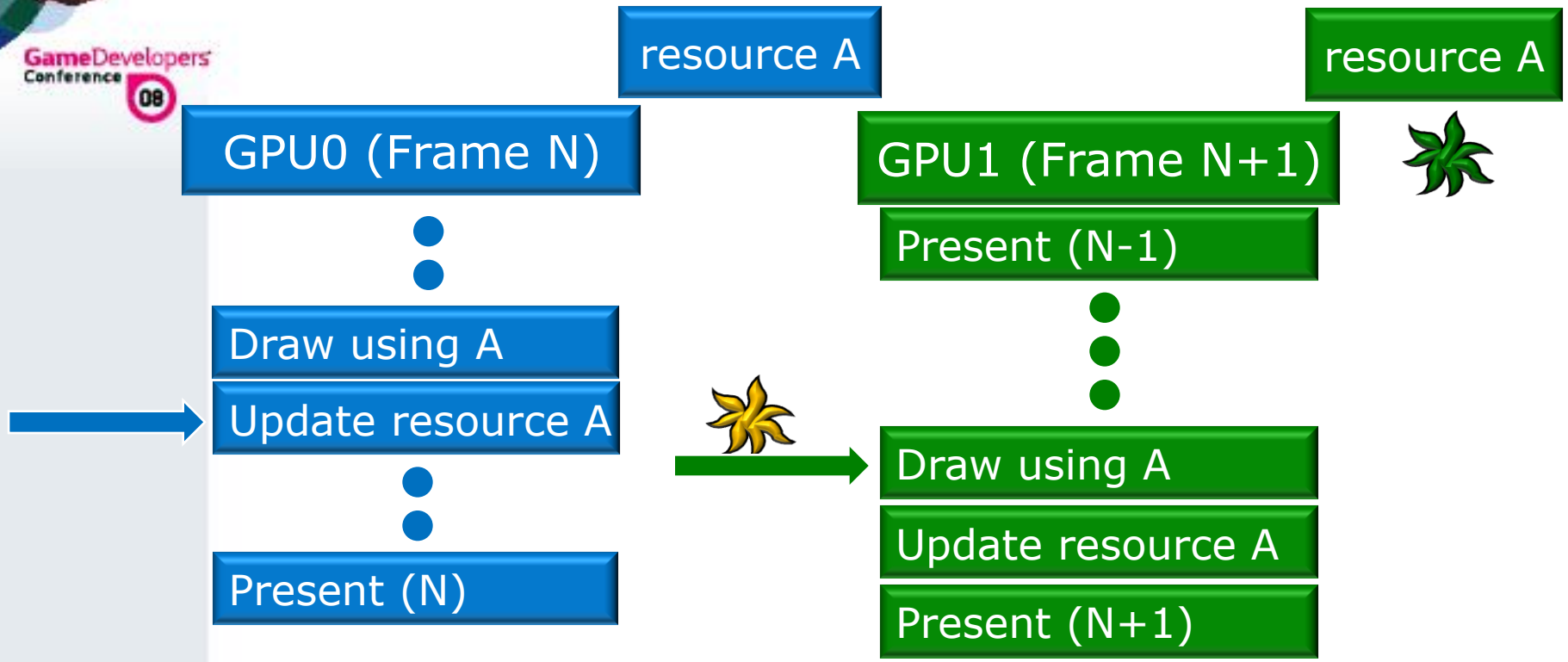
Pitfall: Dependencies Between Frames



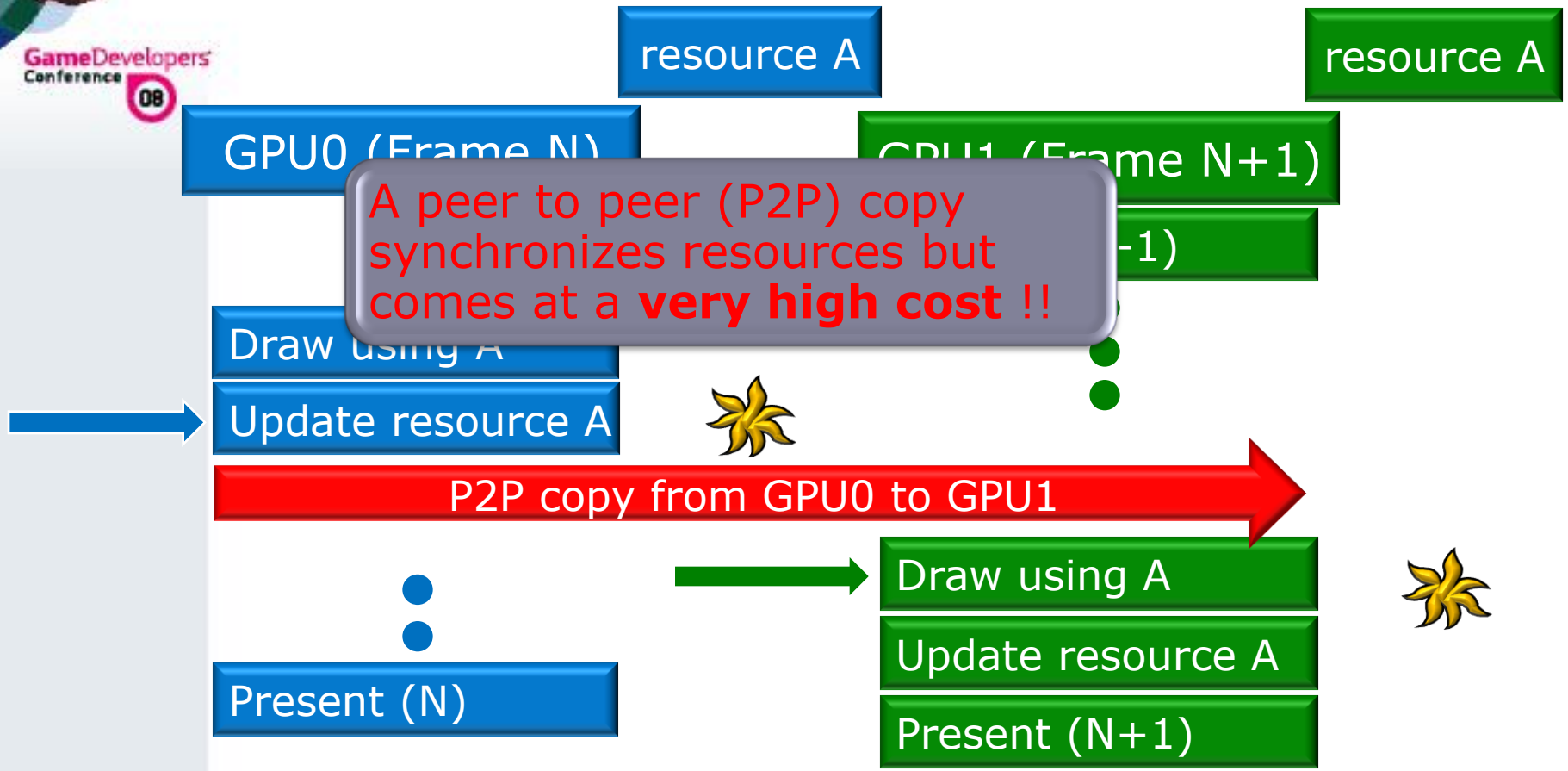
Pitfall: Dependencies Between Frames



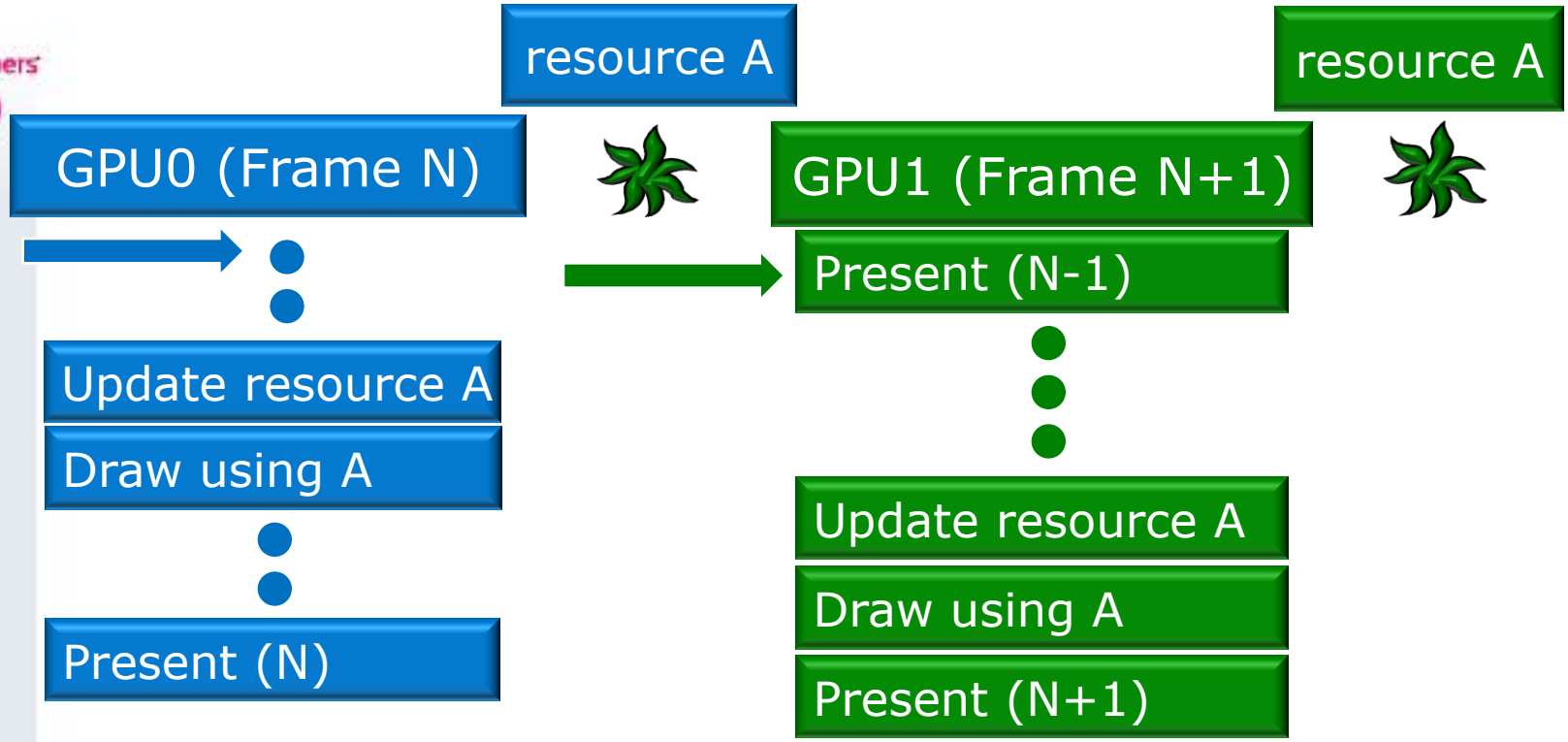
Pitfall: Dependencies Between Frames



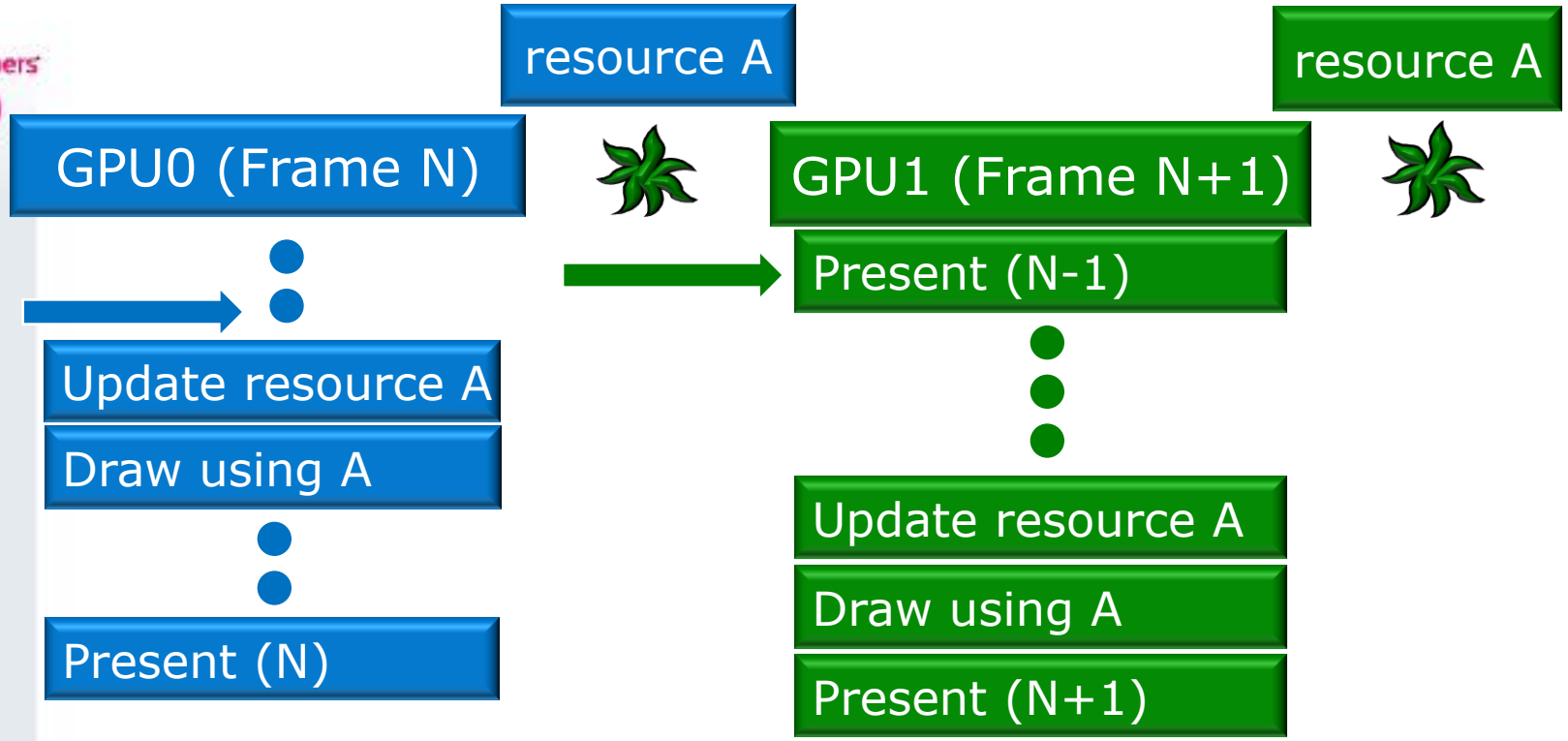
Pitfall: Dependencies Between Frames

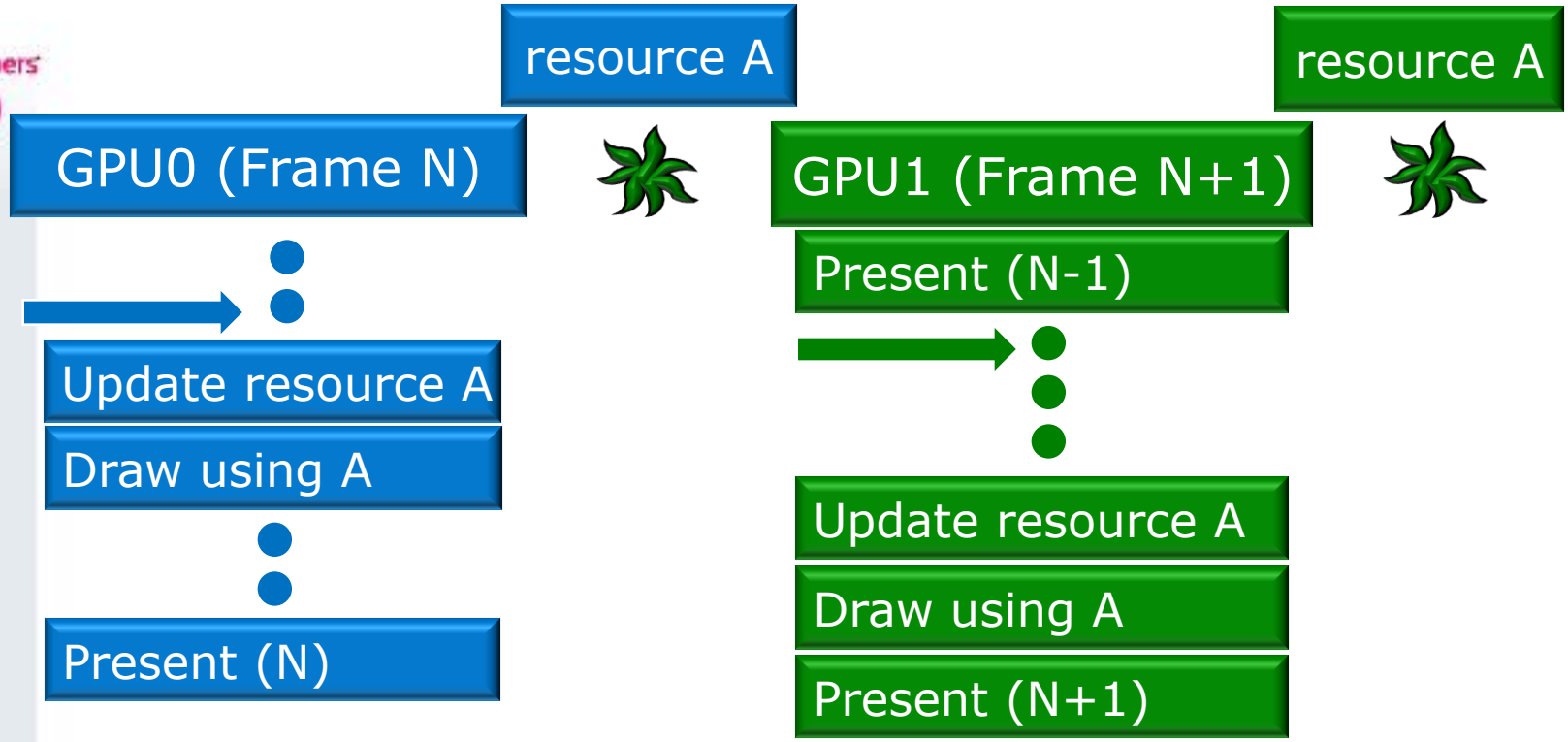


Solution: Resources that Change Every Frame

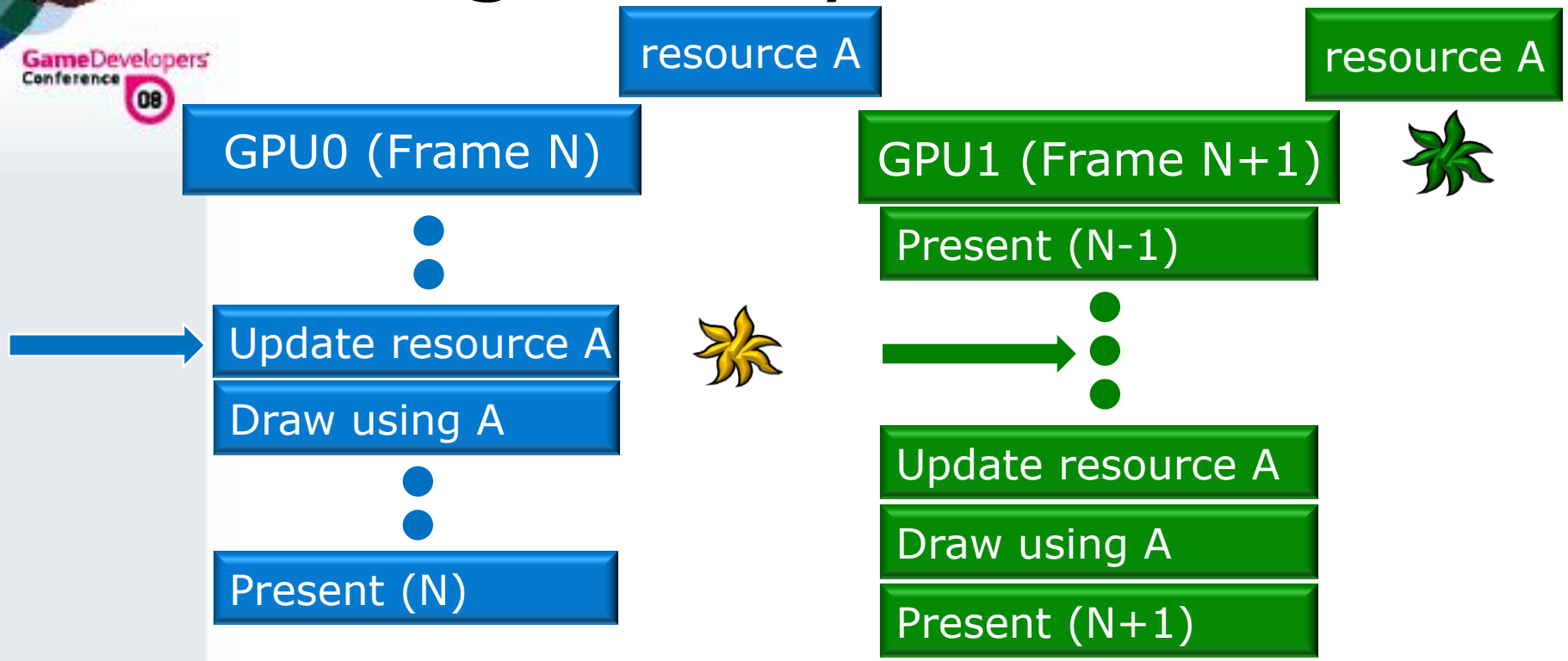


Solution: Resources that Change Every Frame

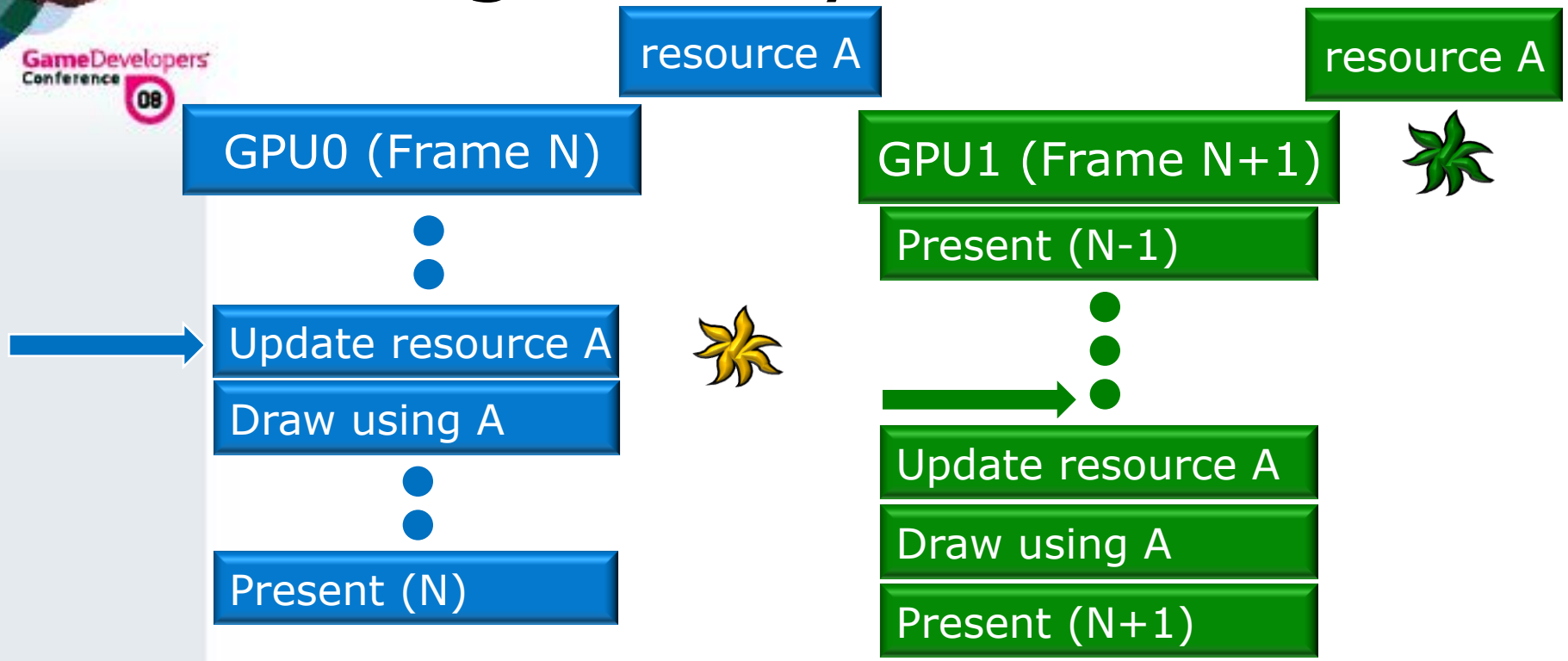



resource A
resource A
GPU0 (Frame N)
GPU1 (Frame N+1)
Present (N-1)
Update resource A
Draw using A
Update resource A
Draw using A
Present (N)
Present (N+1)

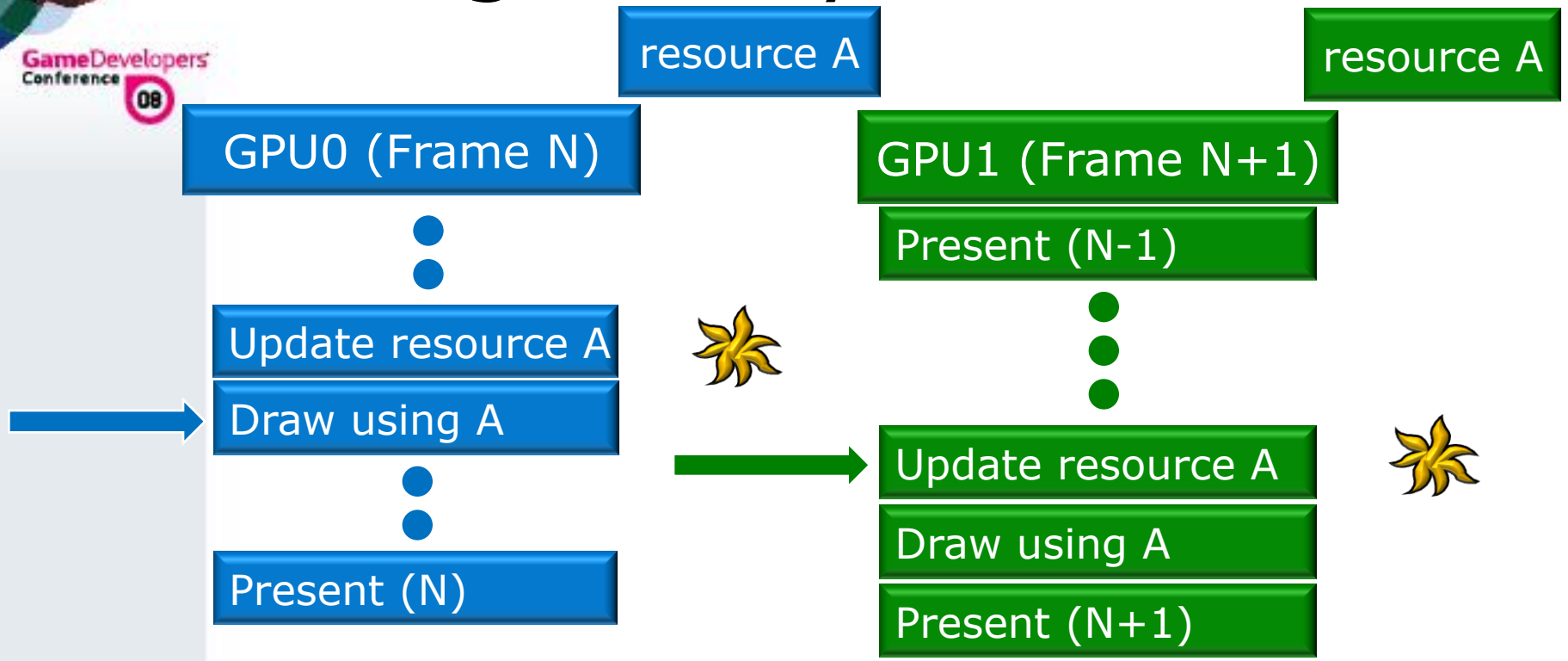
Solution: Resources that Change Every Frame



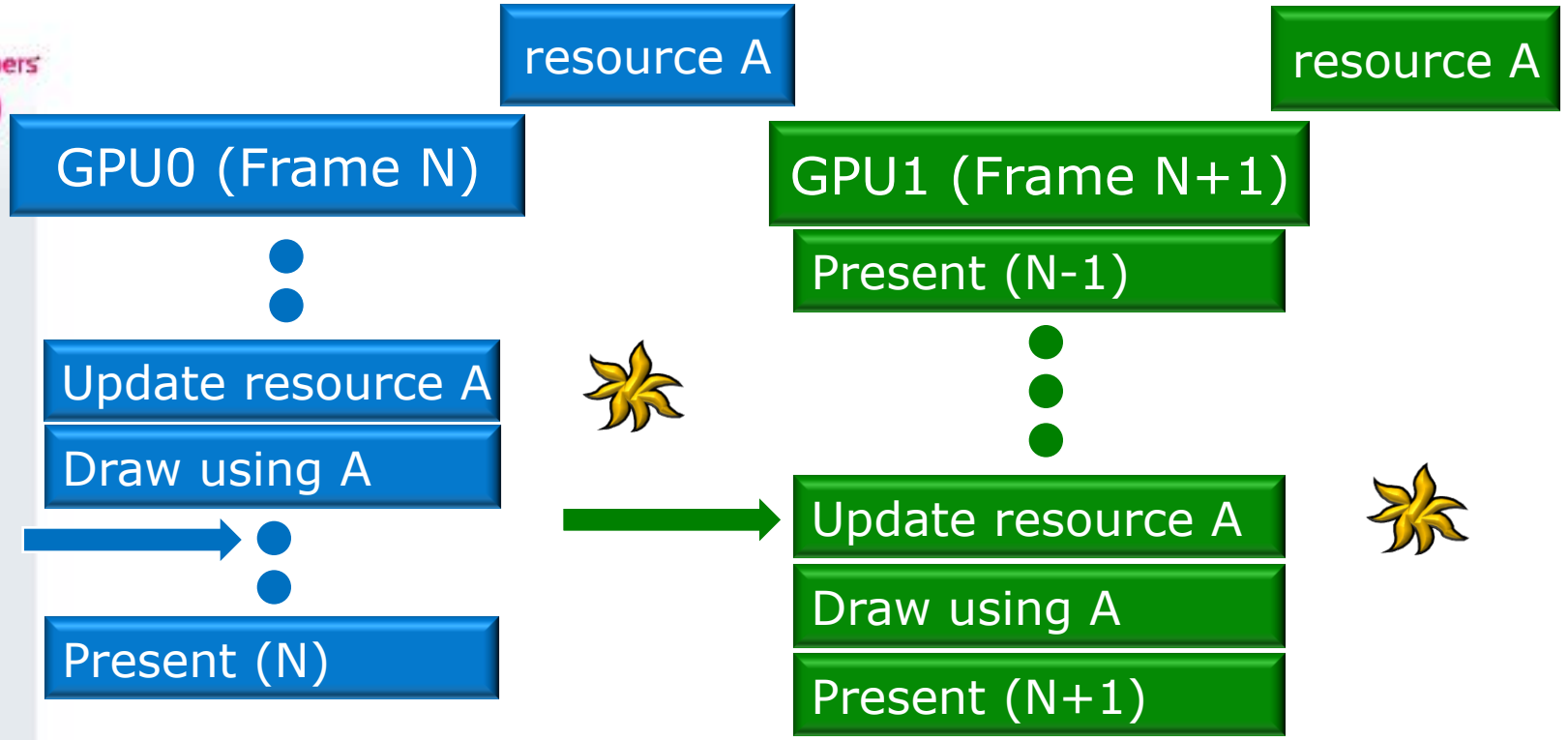
Solution: Resources that Change Every Frame



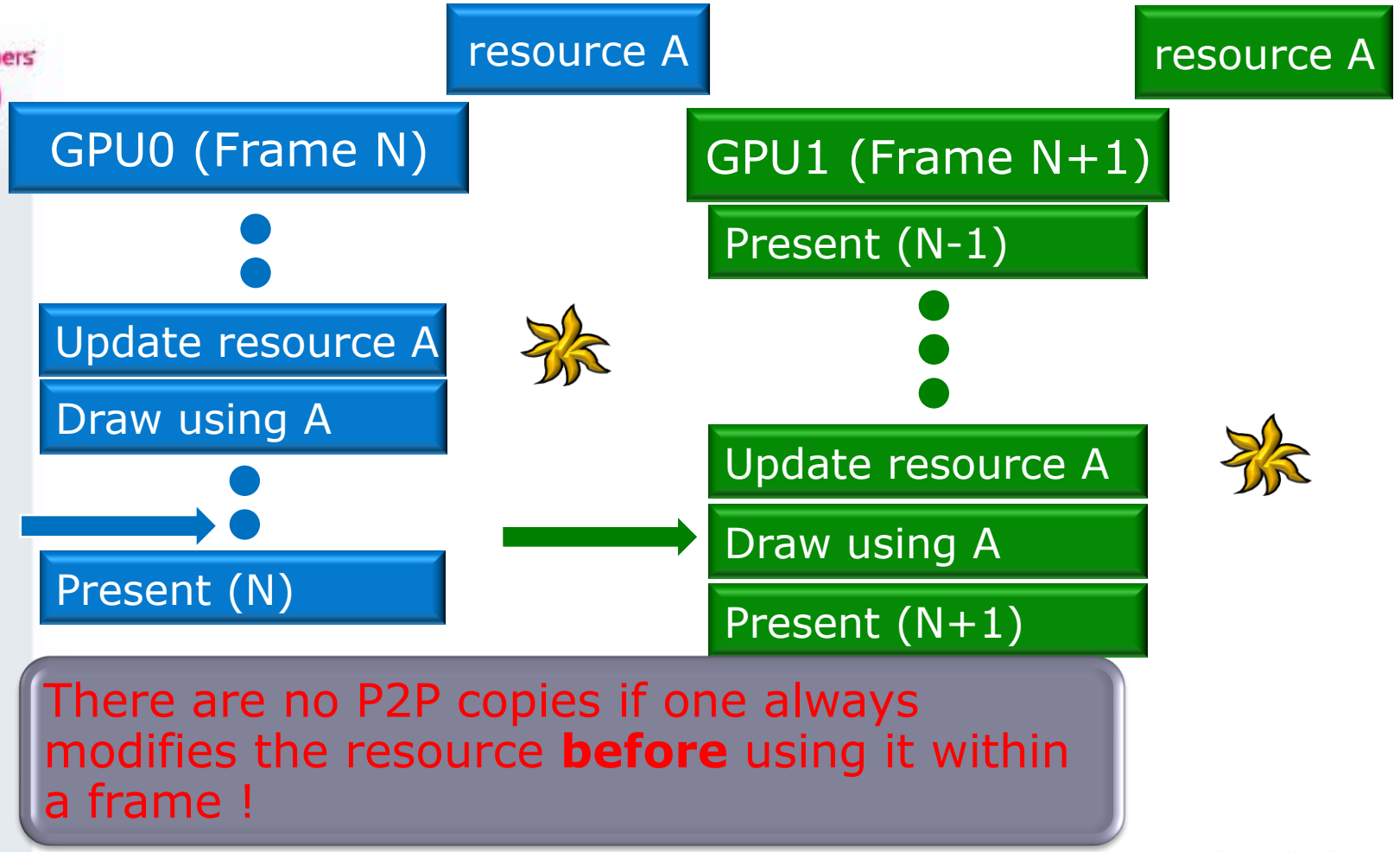
Solution: Resources that Change Every Frame



Solution: Resources that Change Every Frame

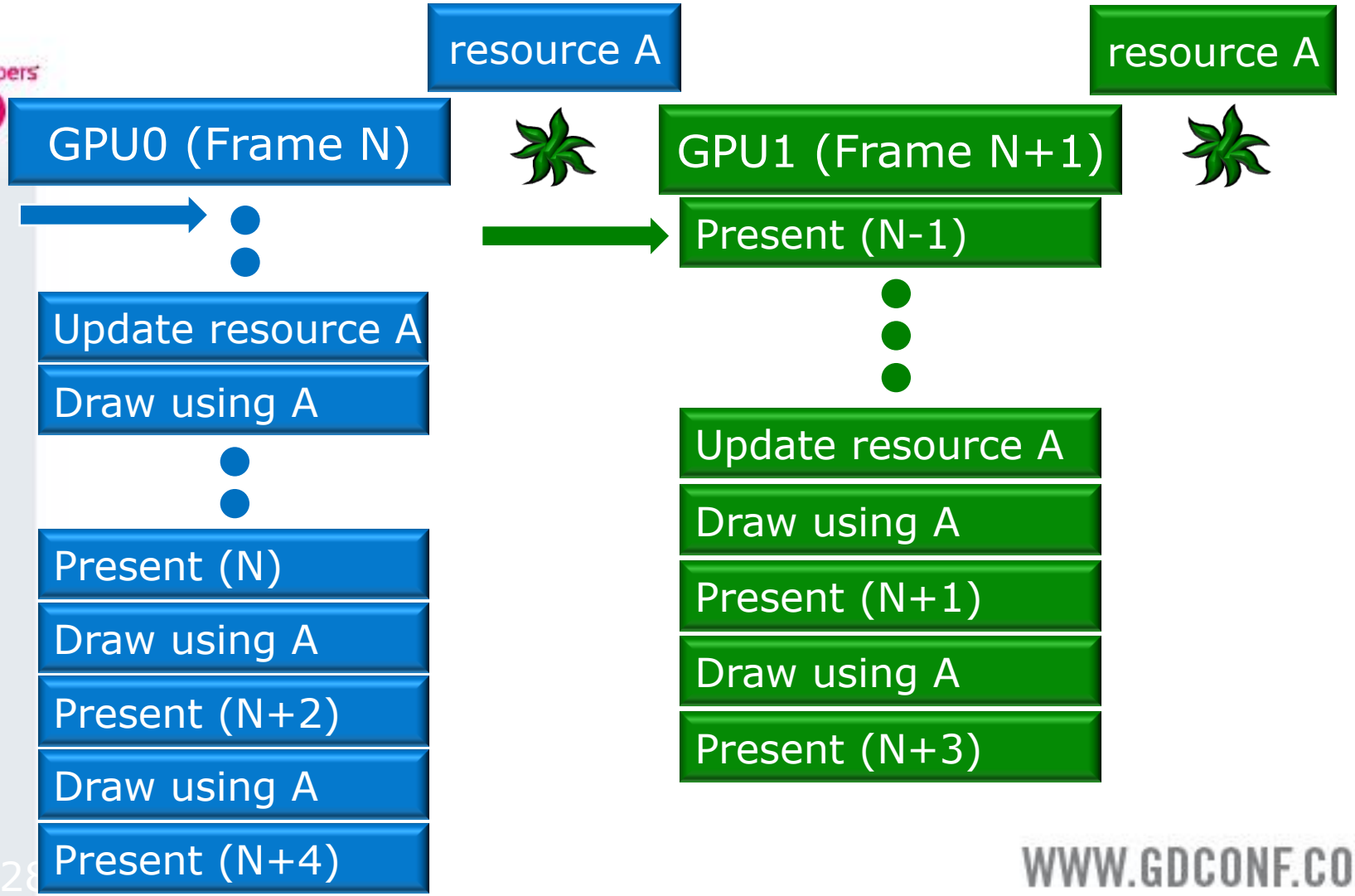


Solution: Resources that Change Every Frame

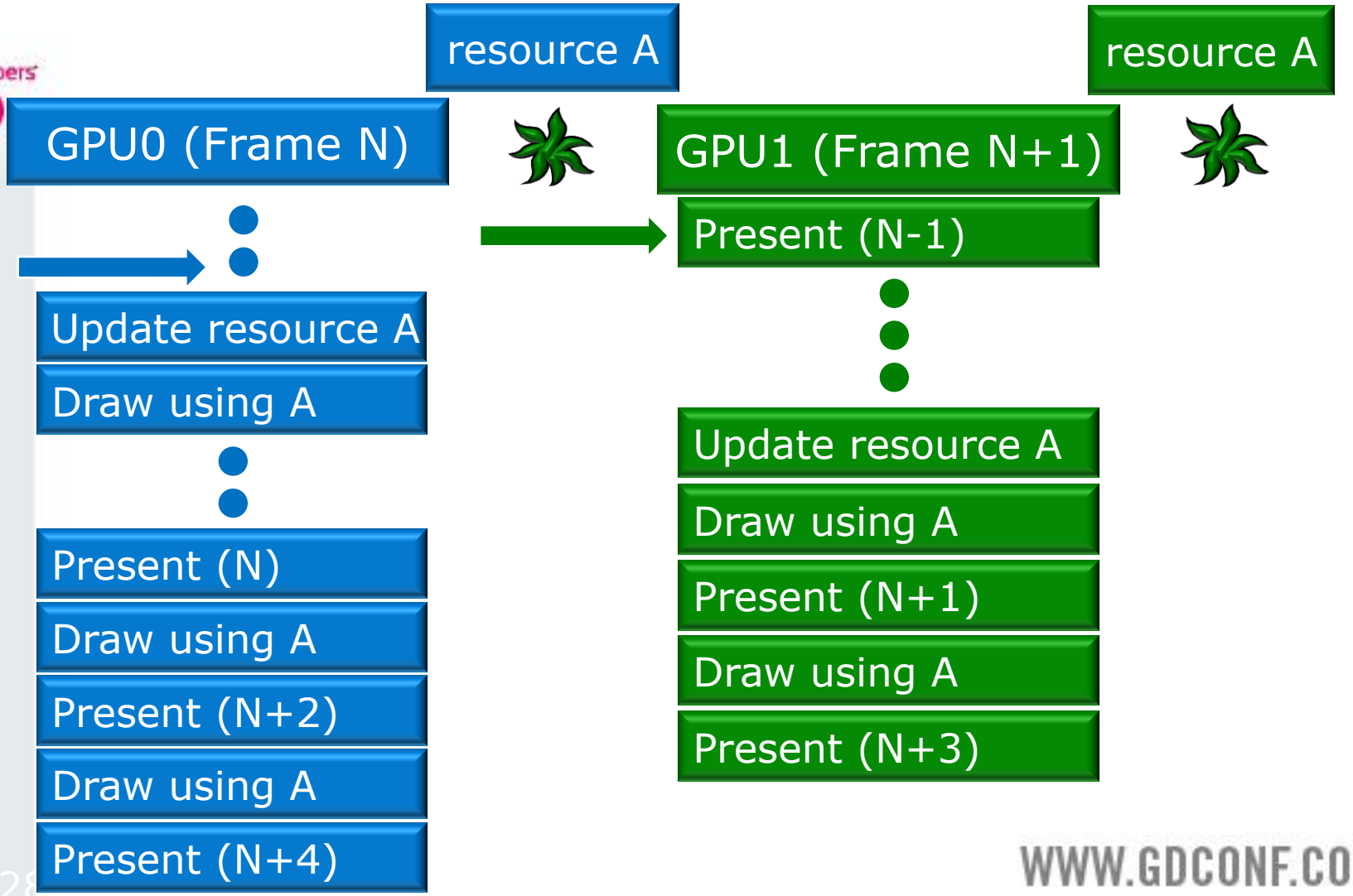


There are no P2P copies if one always modifies the resource **before** using it within a frame !

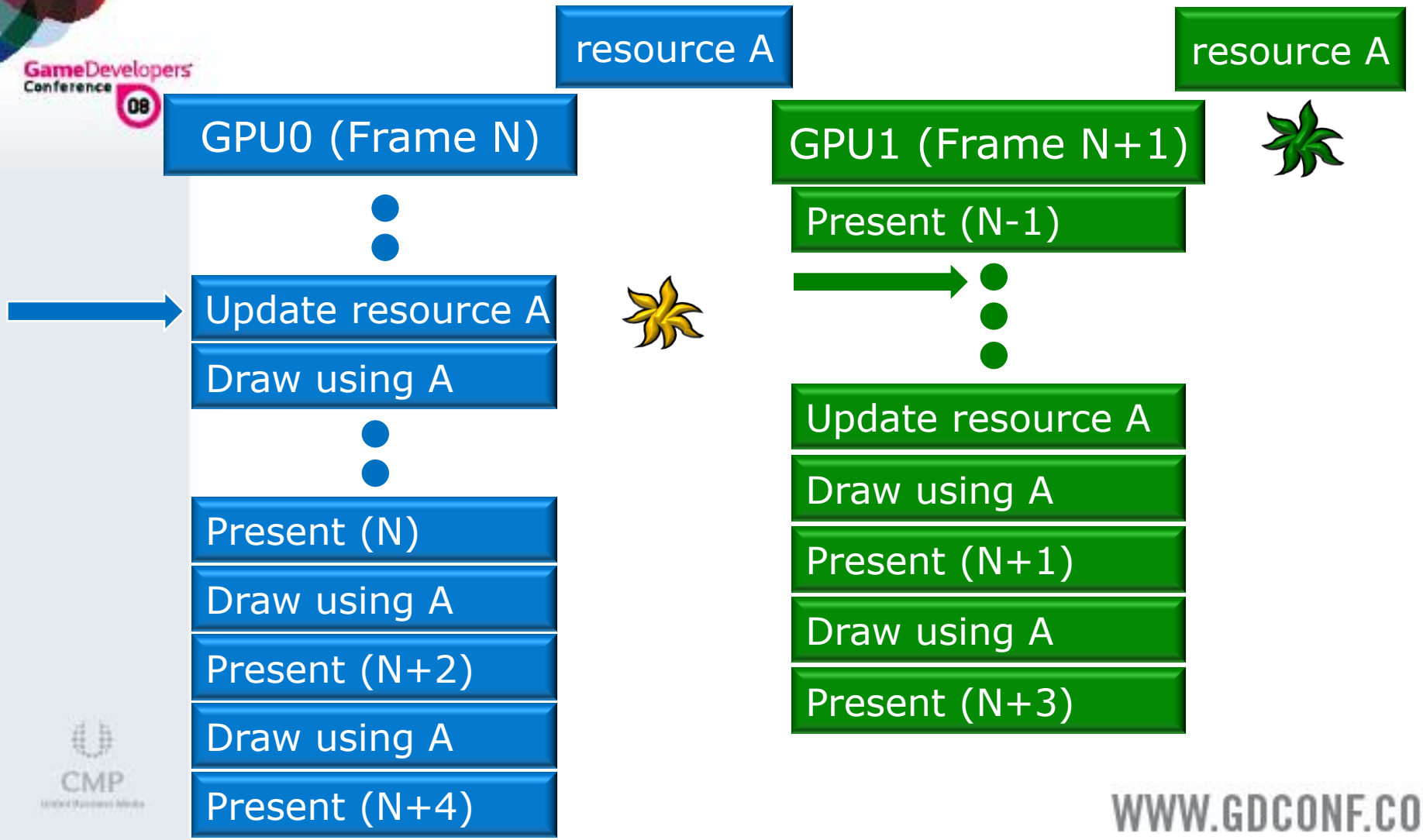
Solution: Resources that Change Every Few Frames



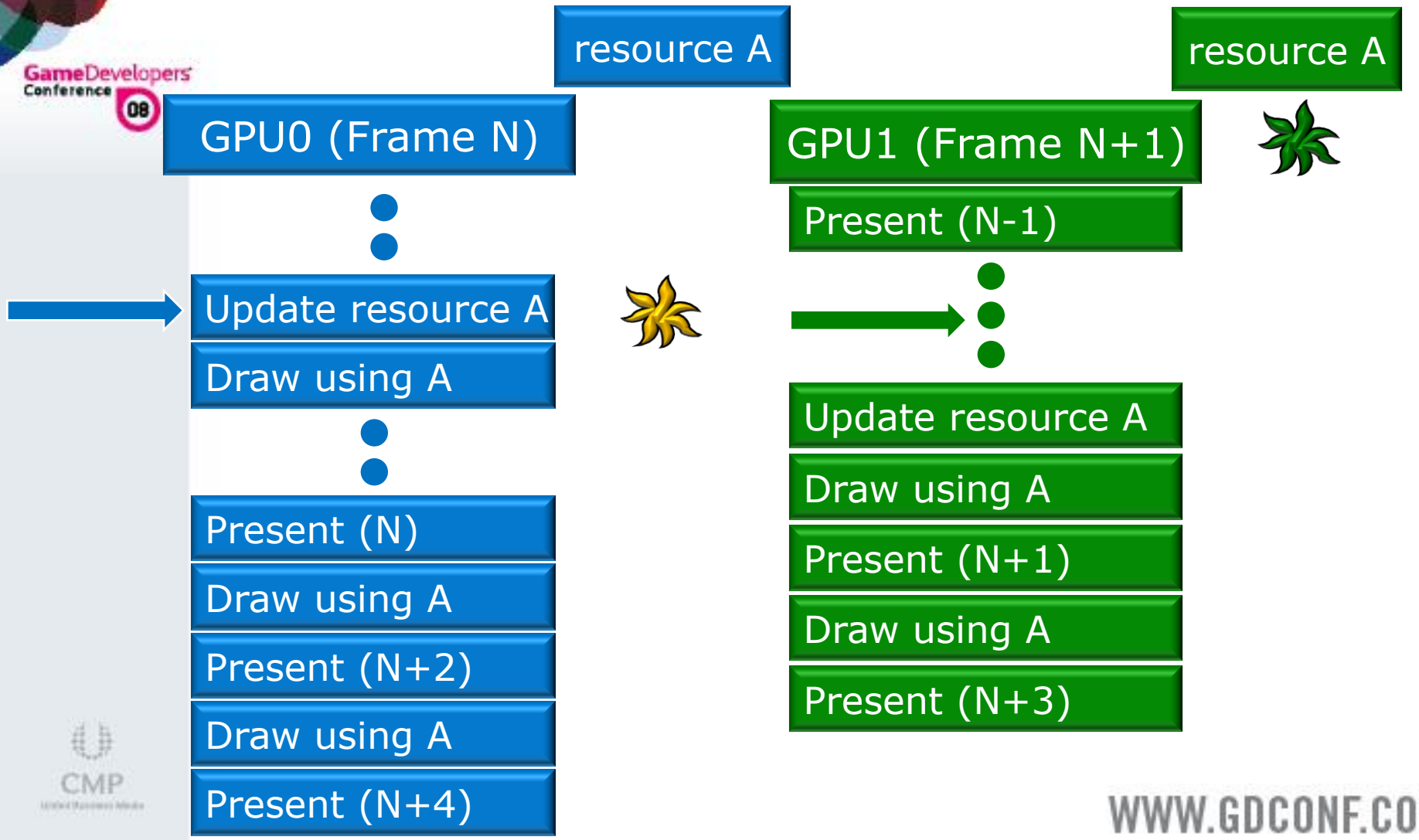
Solution: Resources that Change Every Few Frames



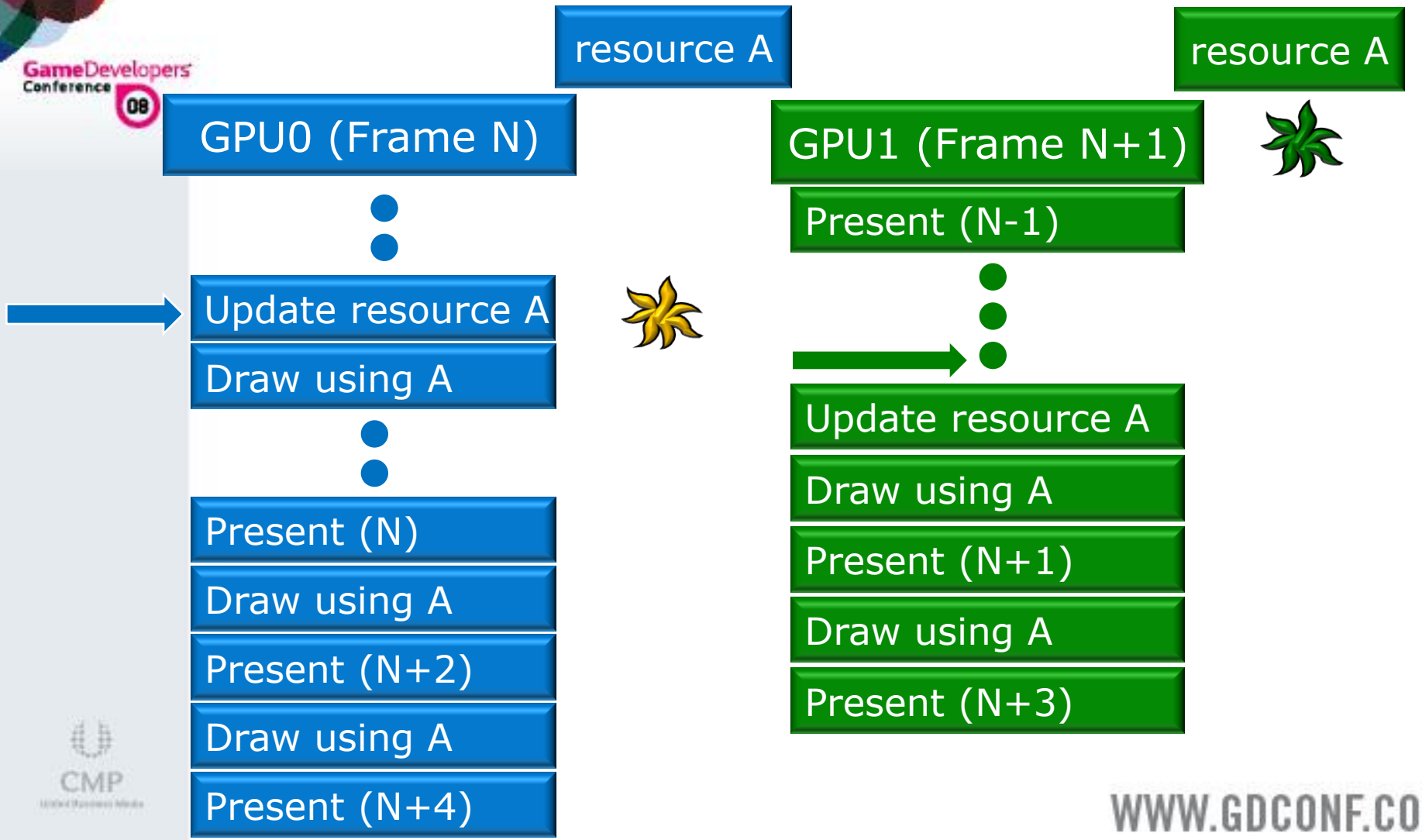
Solution: Resources that Change Every Few Frames



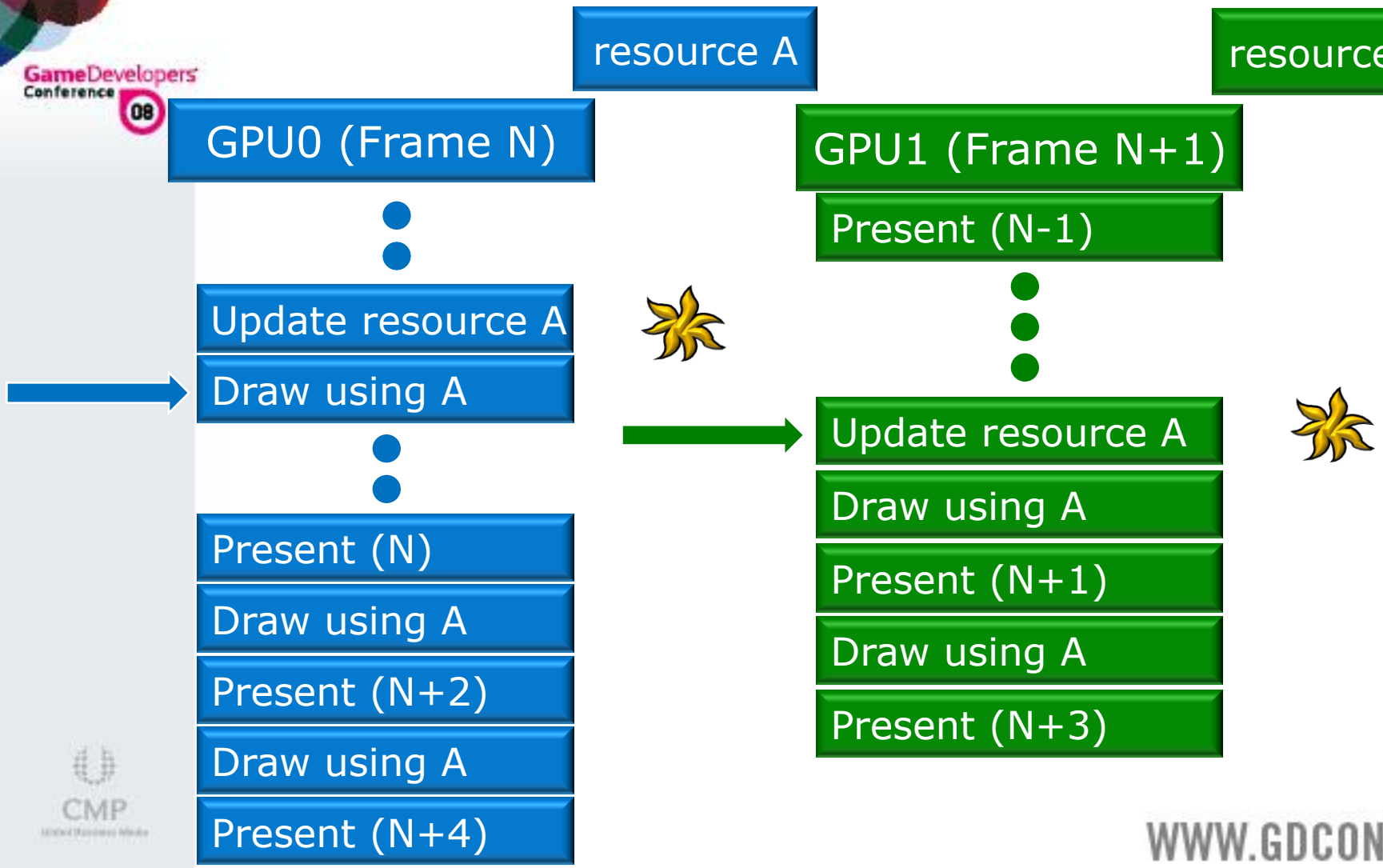
Solution: Resources that Change Every Few Frames



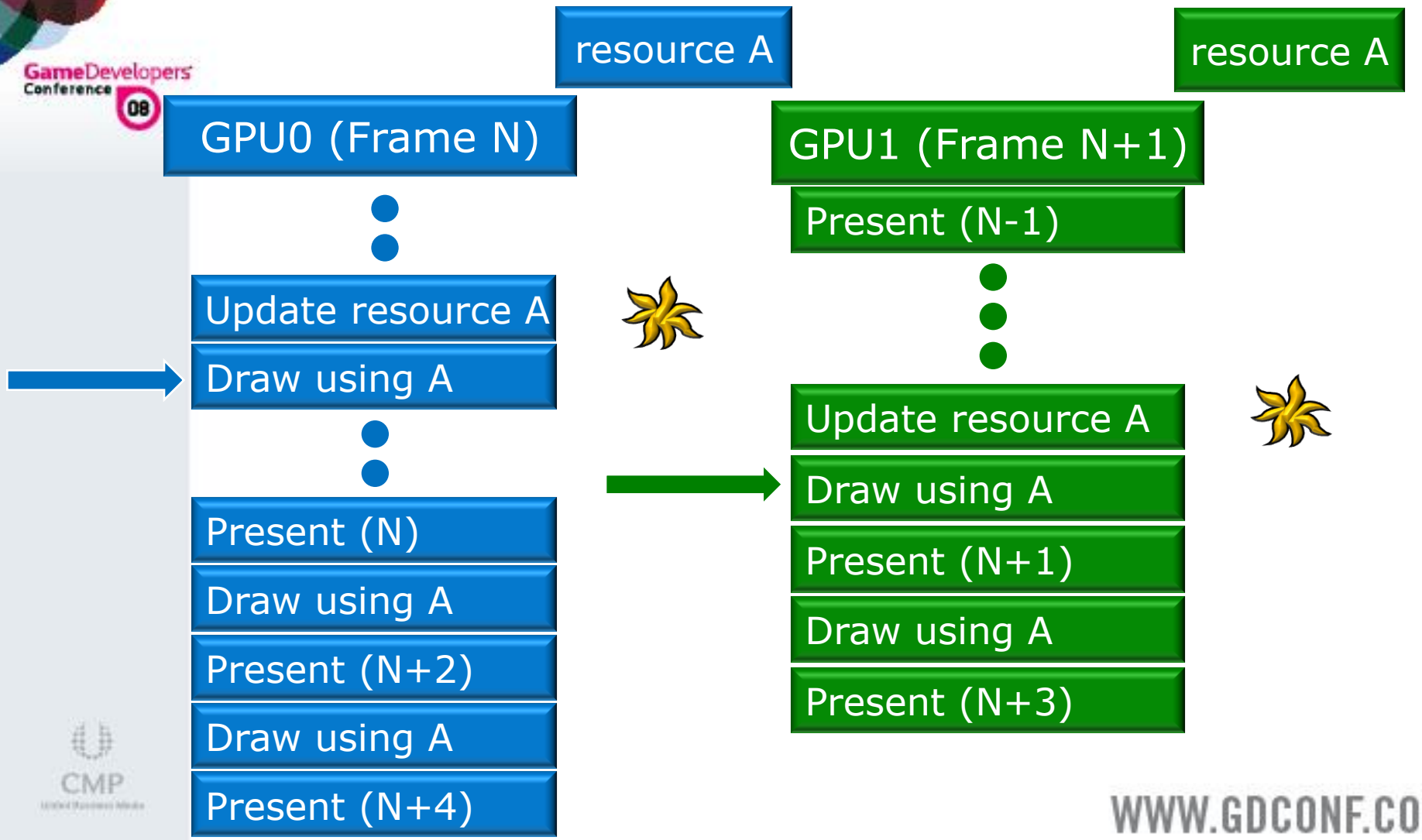
Solution: Resources that Change Every Few Frames



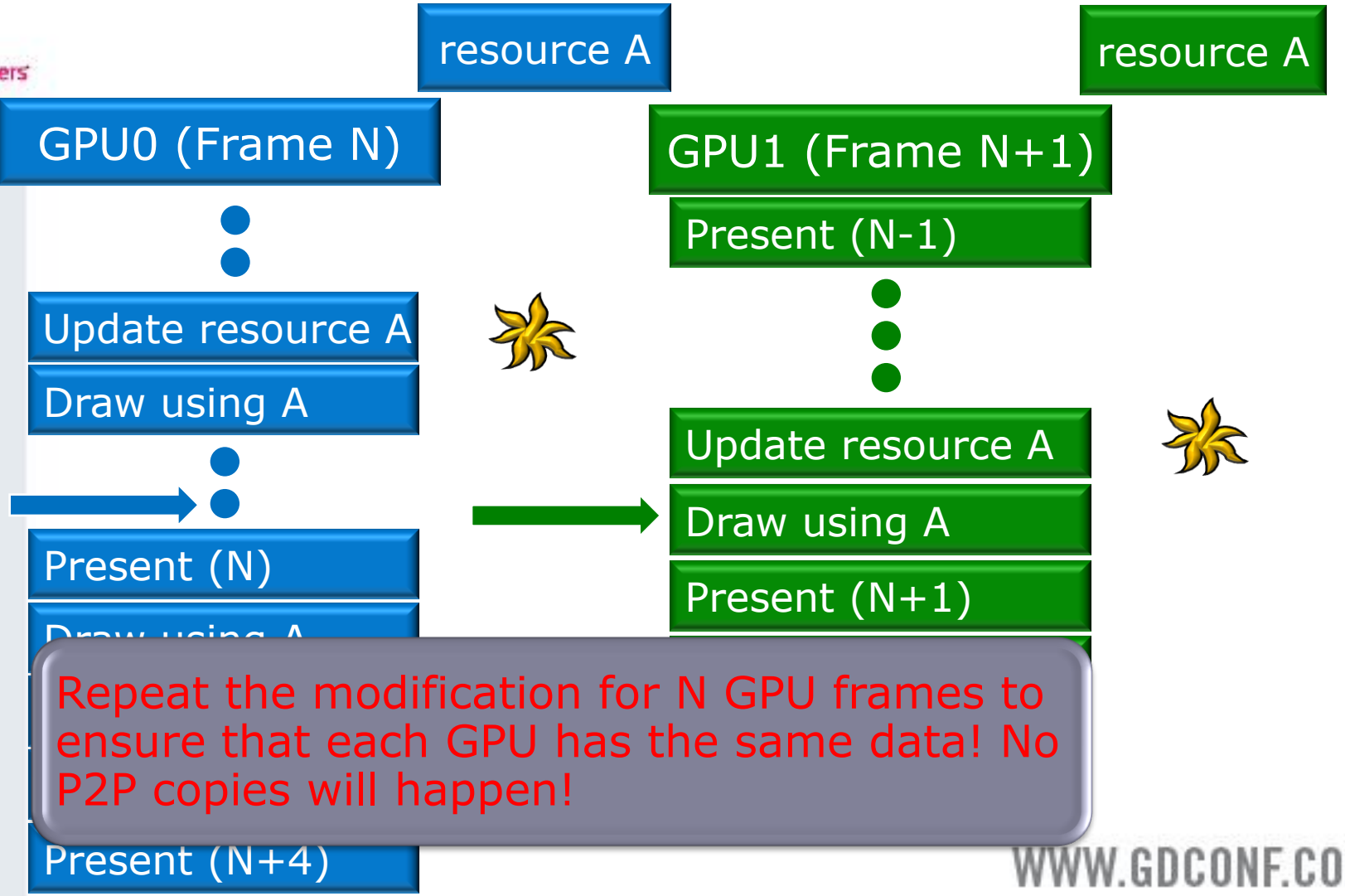
Solution: Resources that Change Every Few Frames



Solution: Resources that Change Every Few Frames



Solution: Resources that Change Every Few Frames



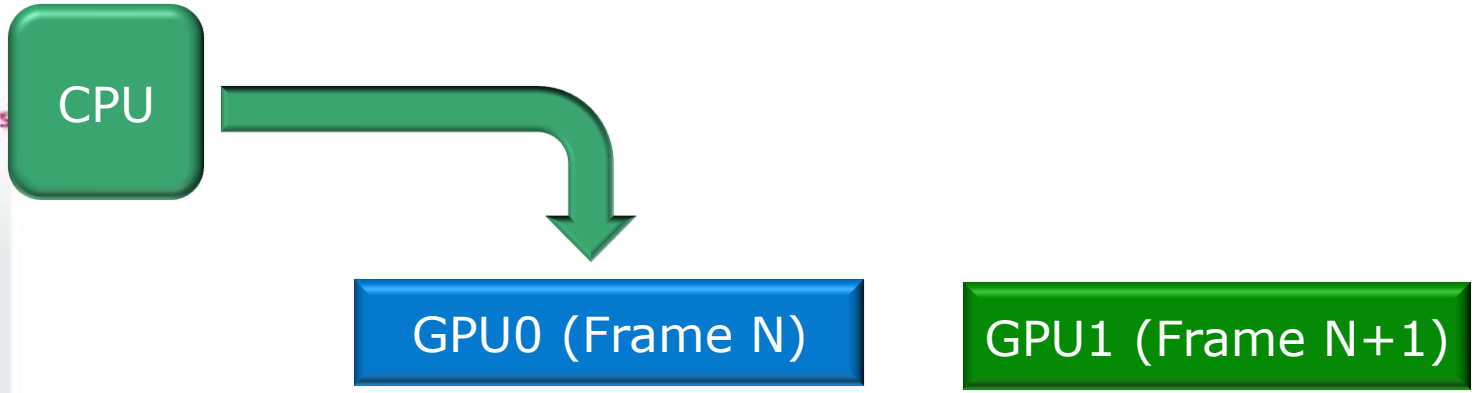
Preventing P2P copies

- NVIDIA: Make sure to Clear RTs that don‘t need to be in sync
- NVIDIA: Use Map(WRITE_DISCARD) to also hint resources that don‘t need to be in sync

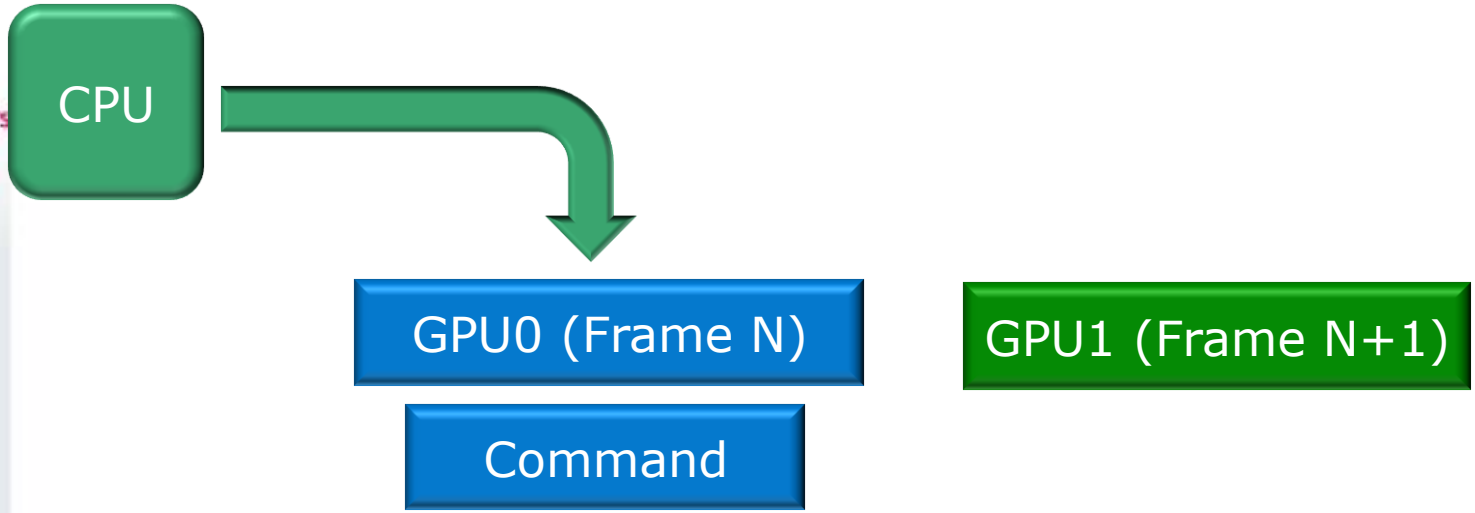
Pitfalls: In DX10 there are Other Ways to Update Resources...

- Drawing to vertex/index buffers
- Stream Out
- CopyResource() calls
- CopySubresourceRegion() calls
- GenerateMips() calls
- ResolveSubresource() calls
- Do not use same resource as destination of both Map(WRITE_DISCARD) and CopyResource/CopySubresourceRegion calls

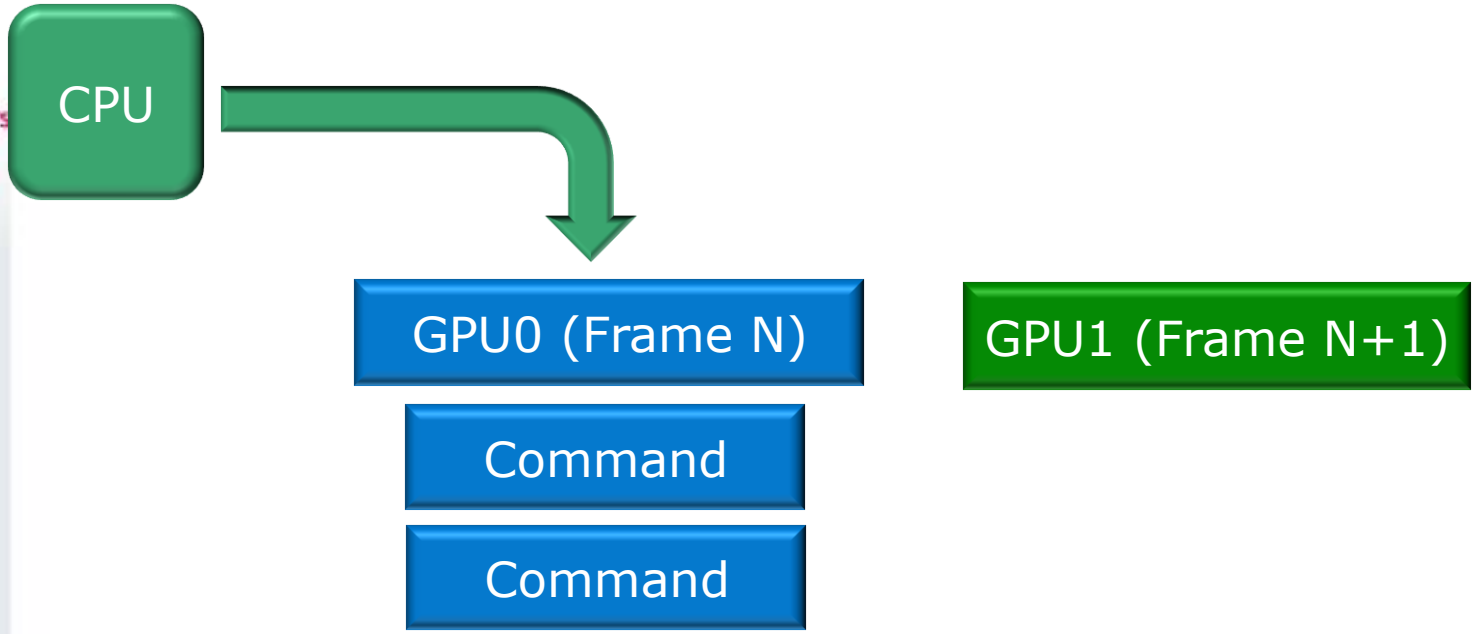
Pitfall: Waiting on Queries



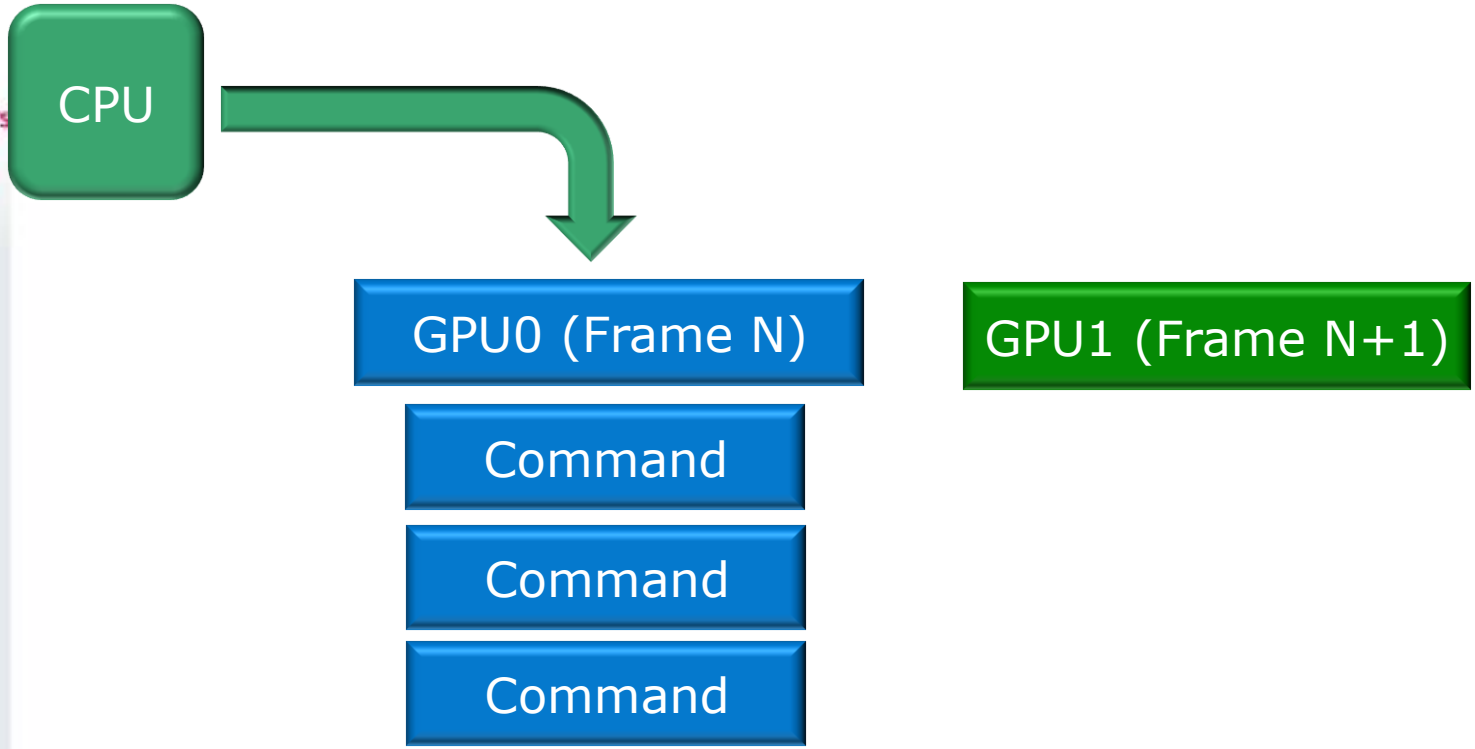
Pitfall: Waiting on Queries



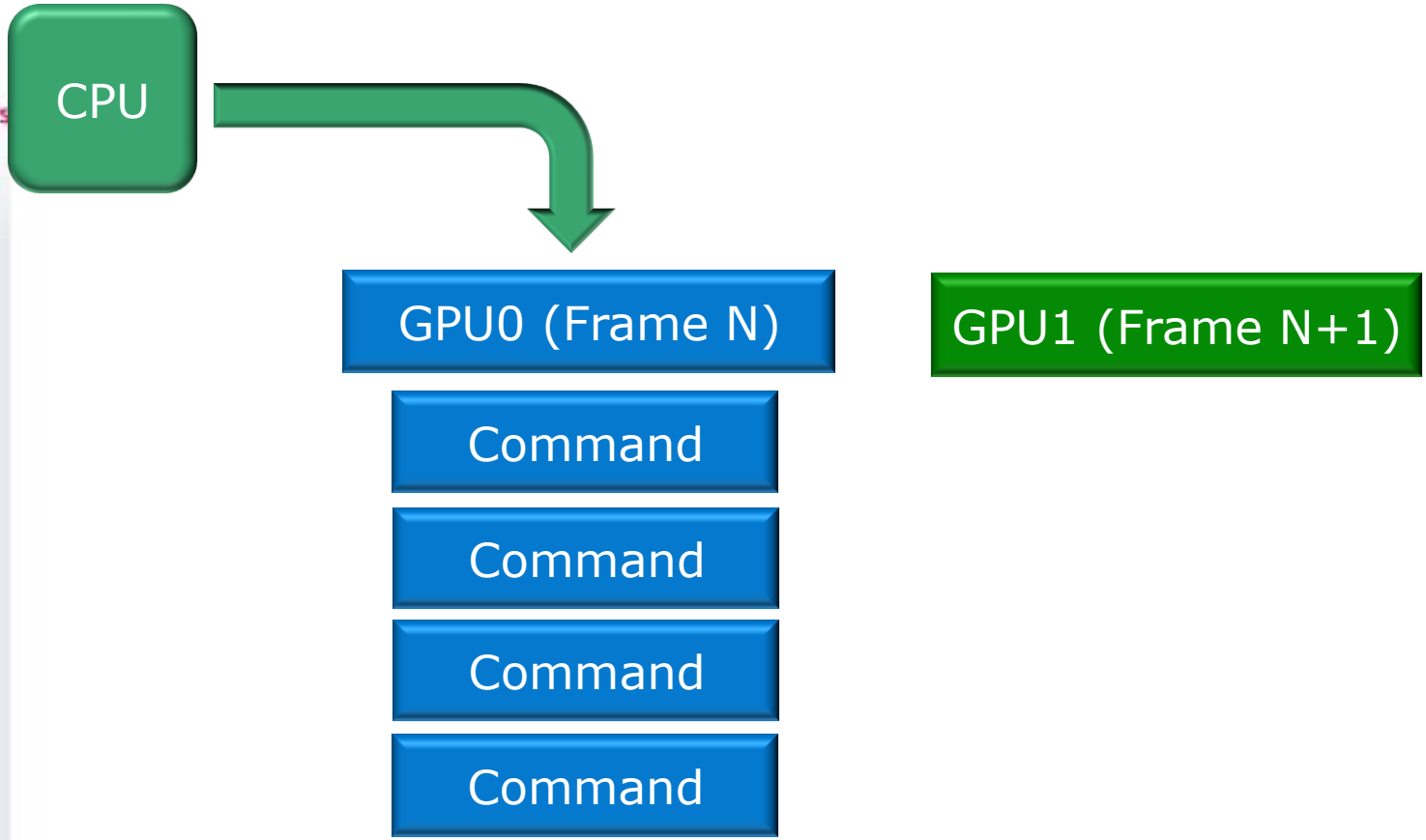
Pitfall: Waiting on Queries



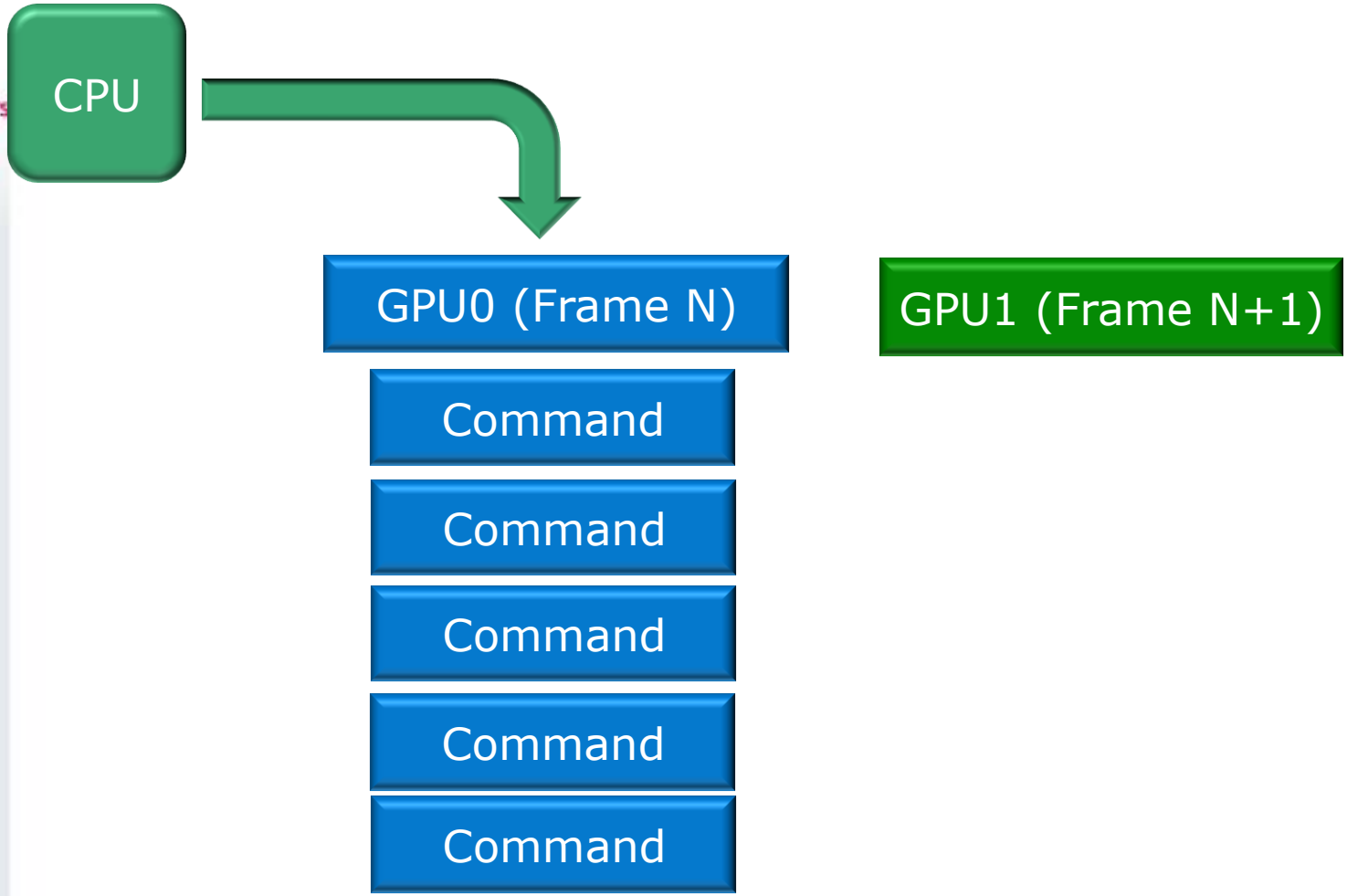
Pitfall: Waiting on Queries



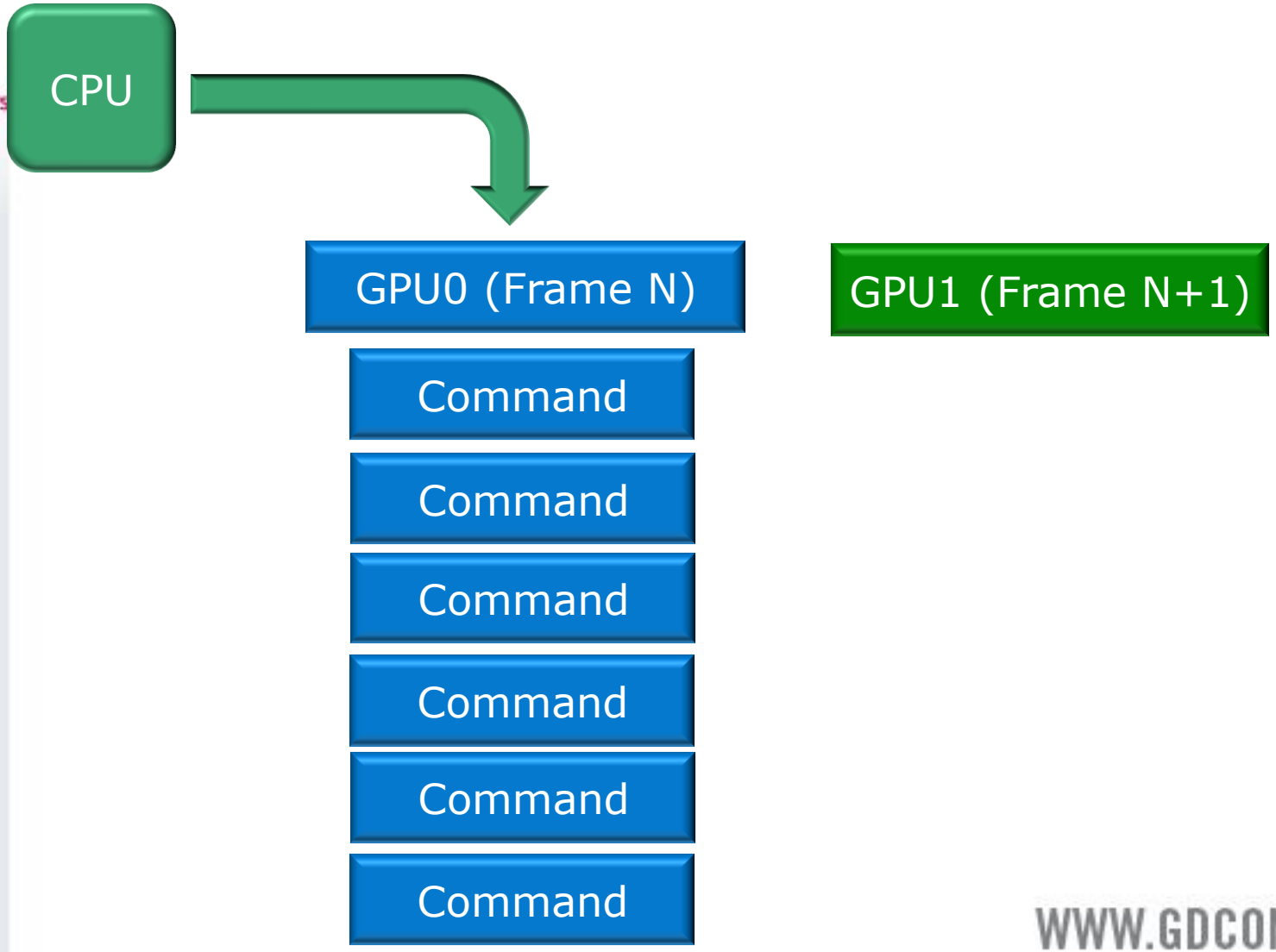
Pitfall: Waiting on Queries



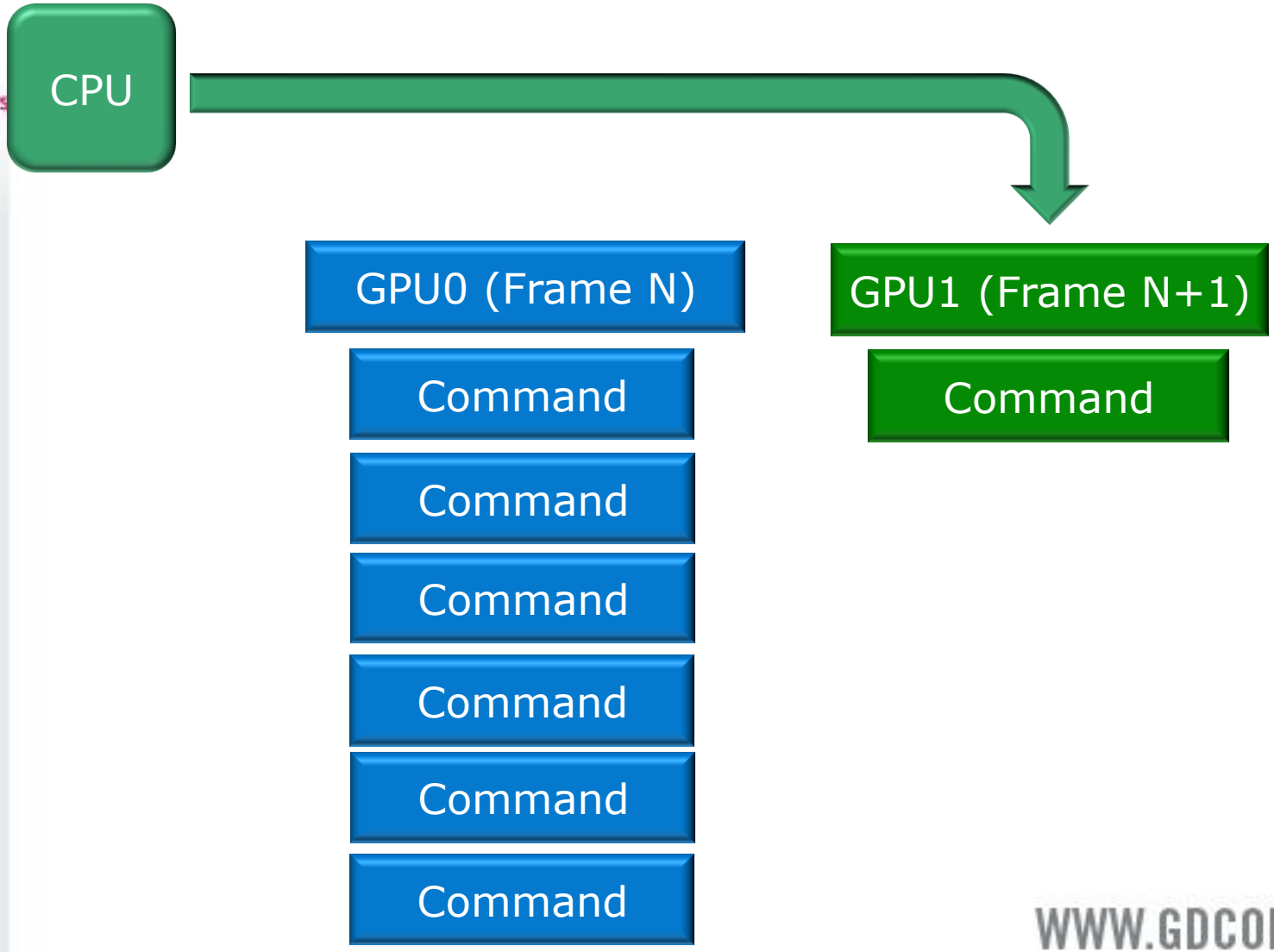
Pitfall: Waiting on Queries



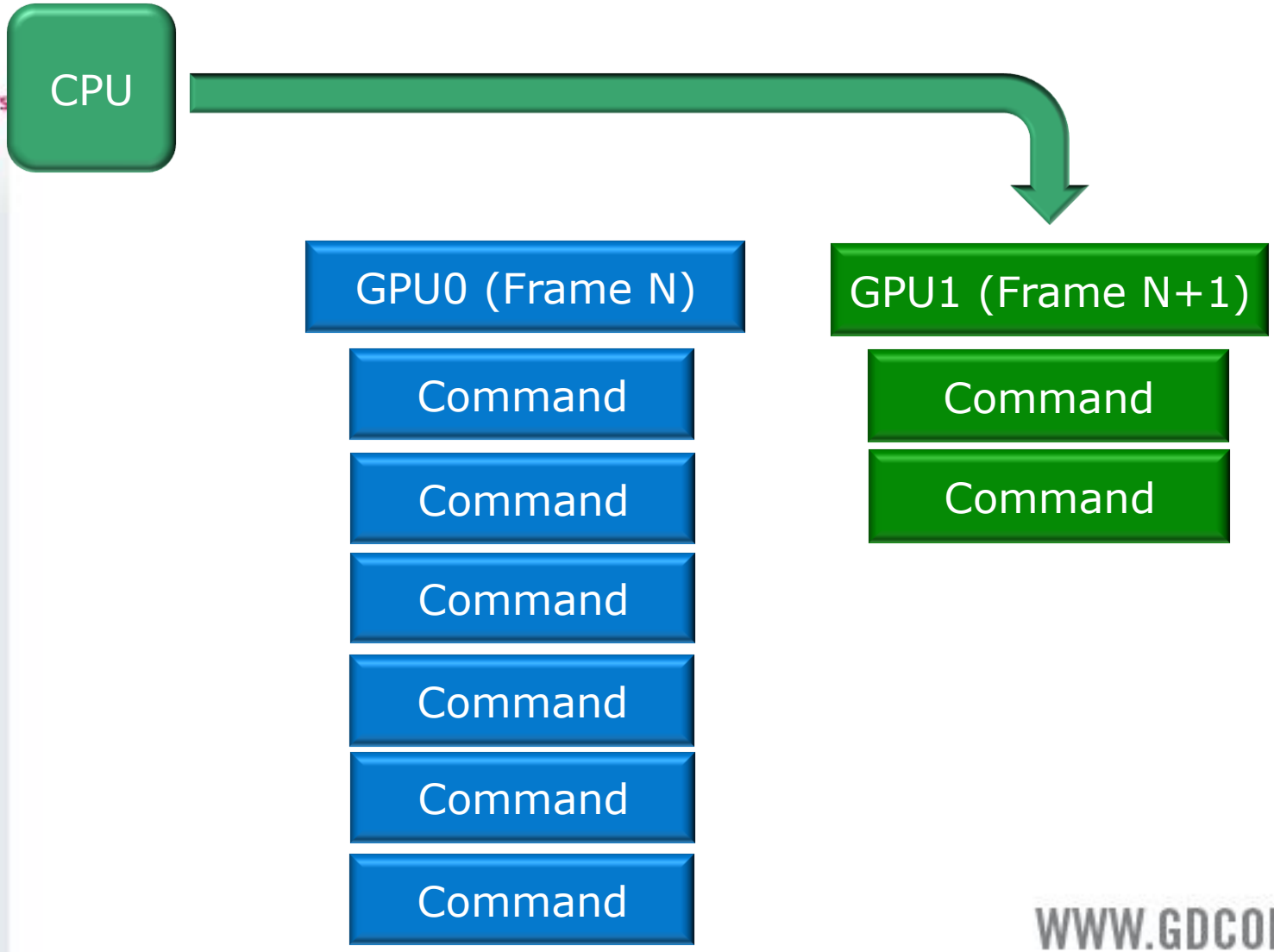
Pitfall: Waiting on Queries



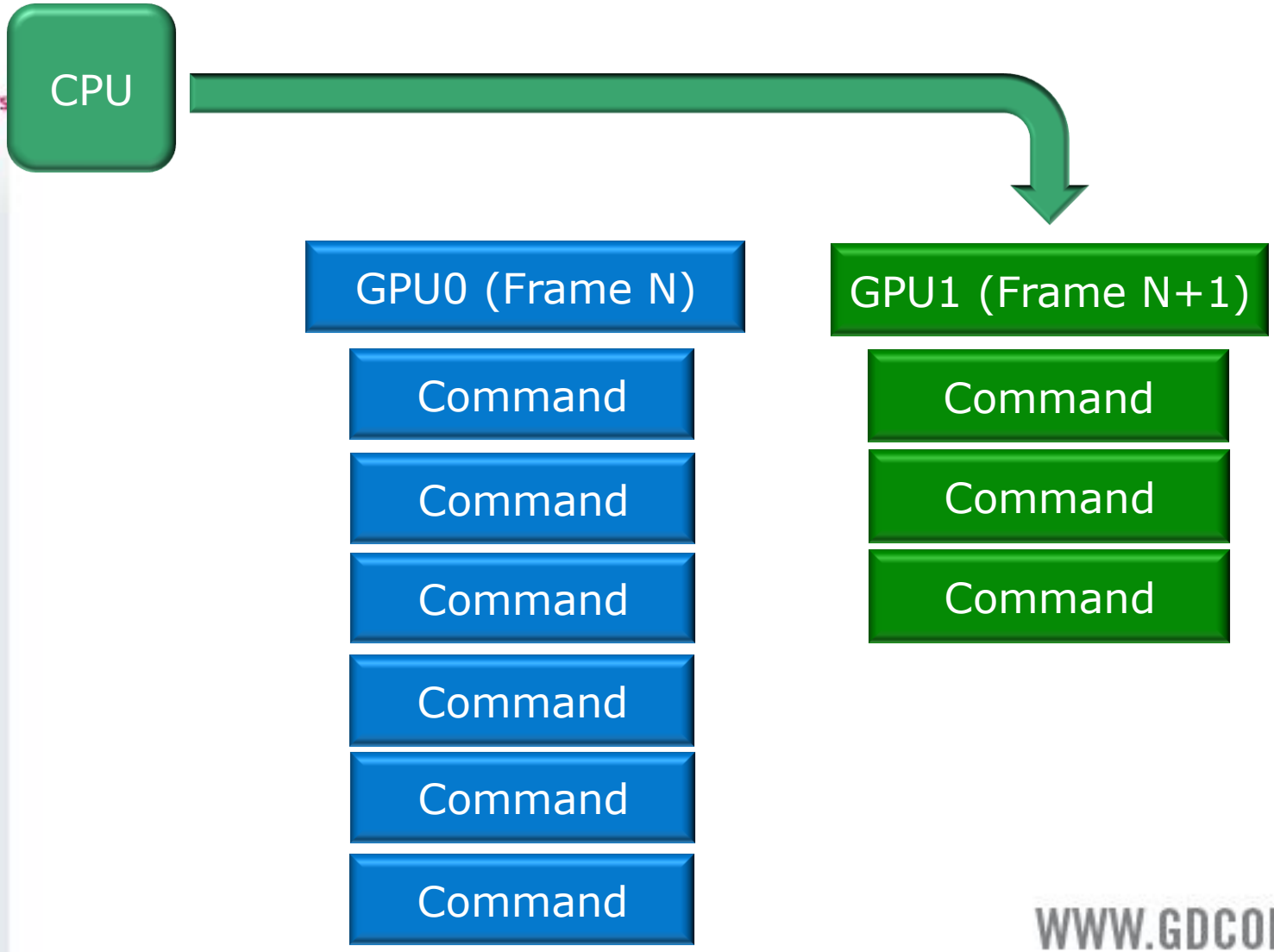
Pitfall: Waiting on Queries



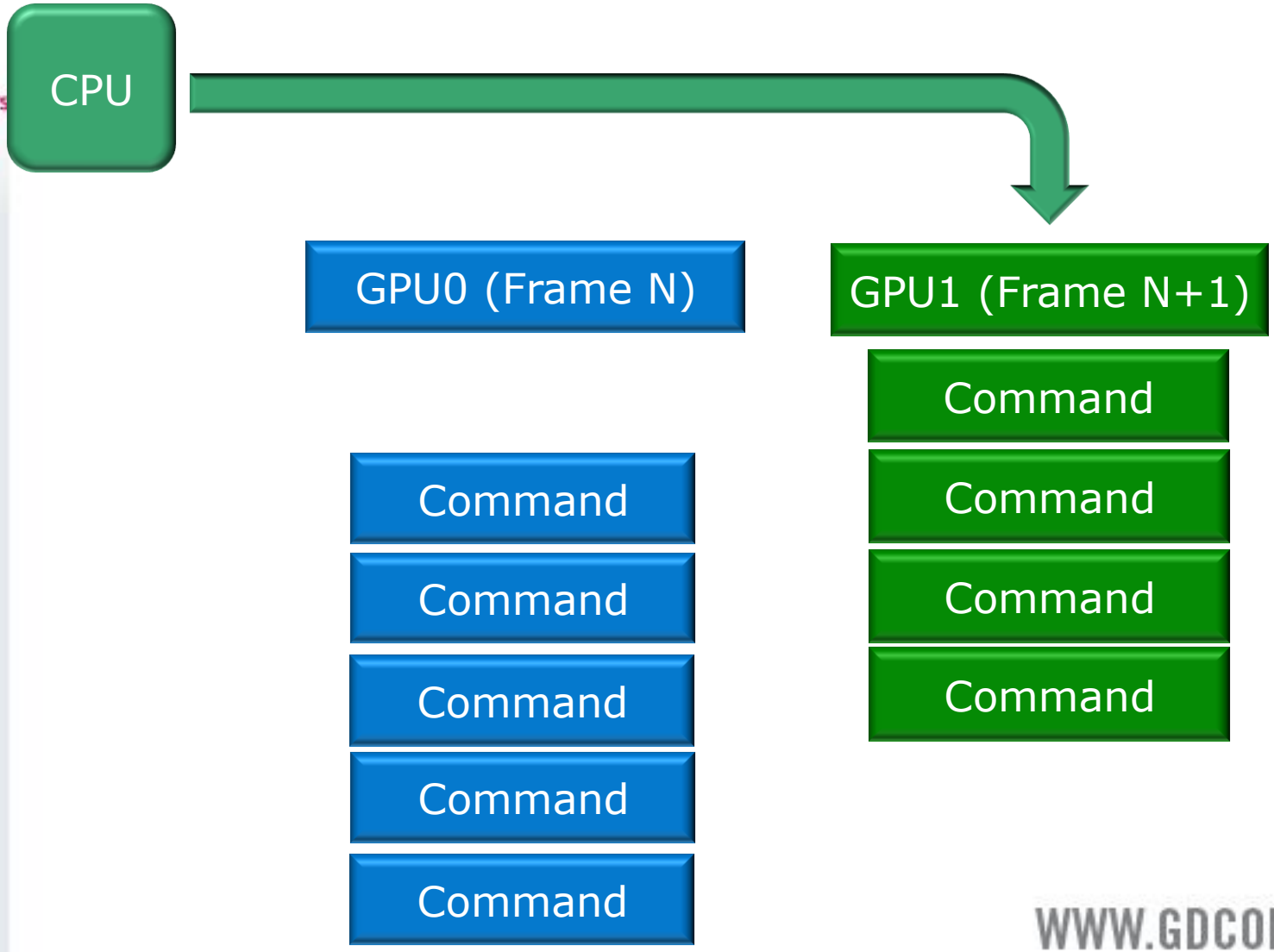
Pitfall: Waiting on Queries



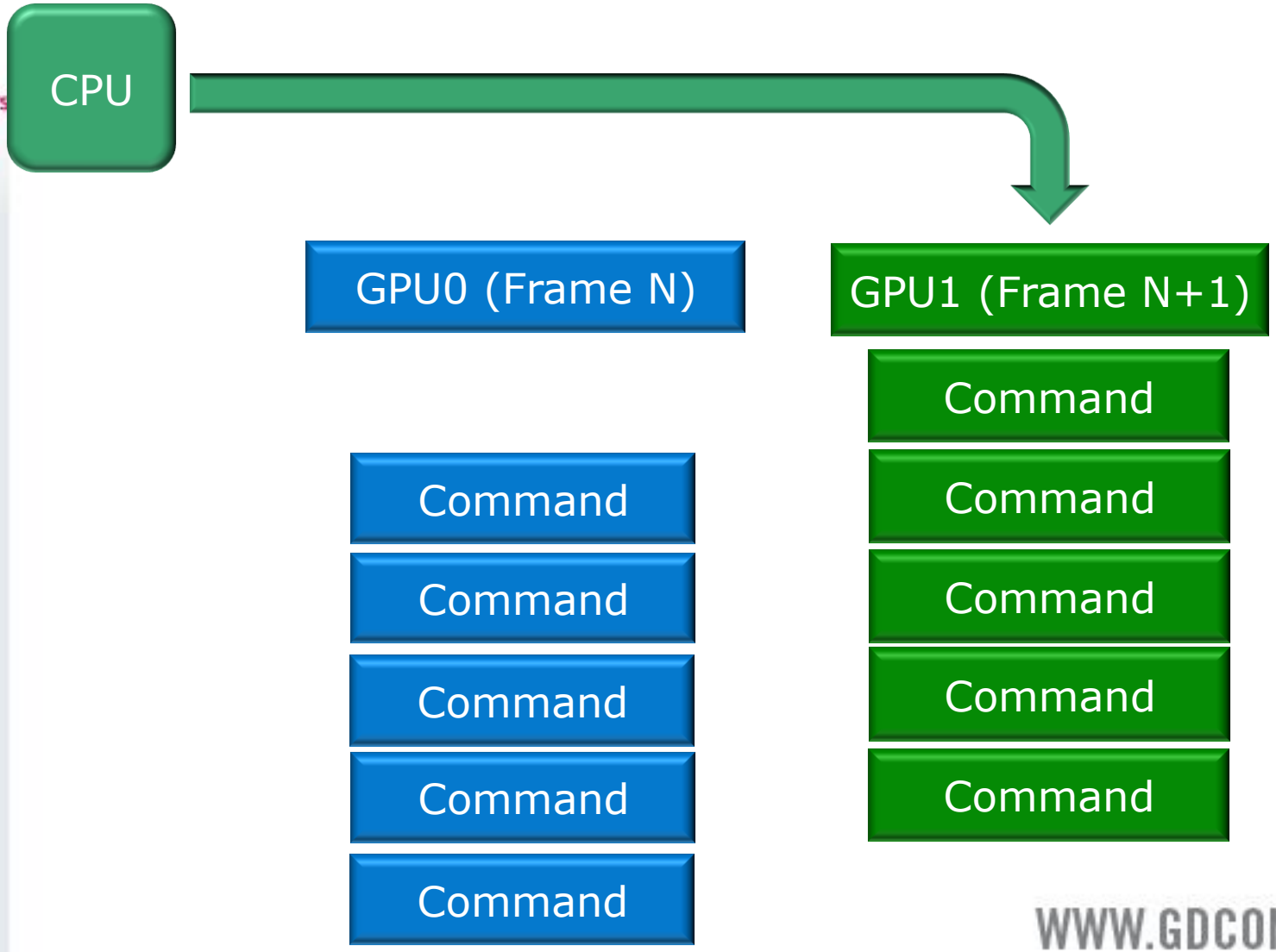
Pitfall: Waiting on Queries



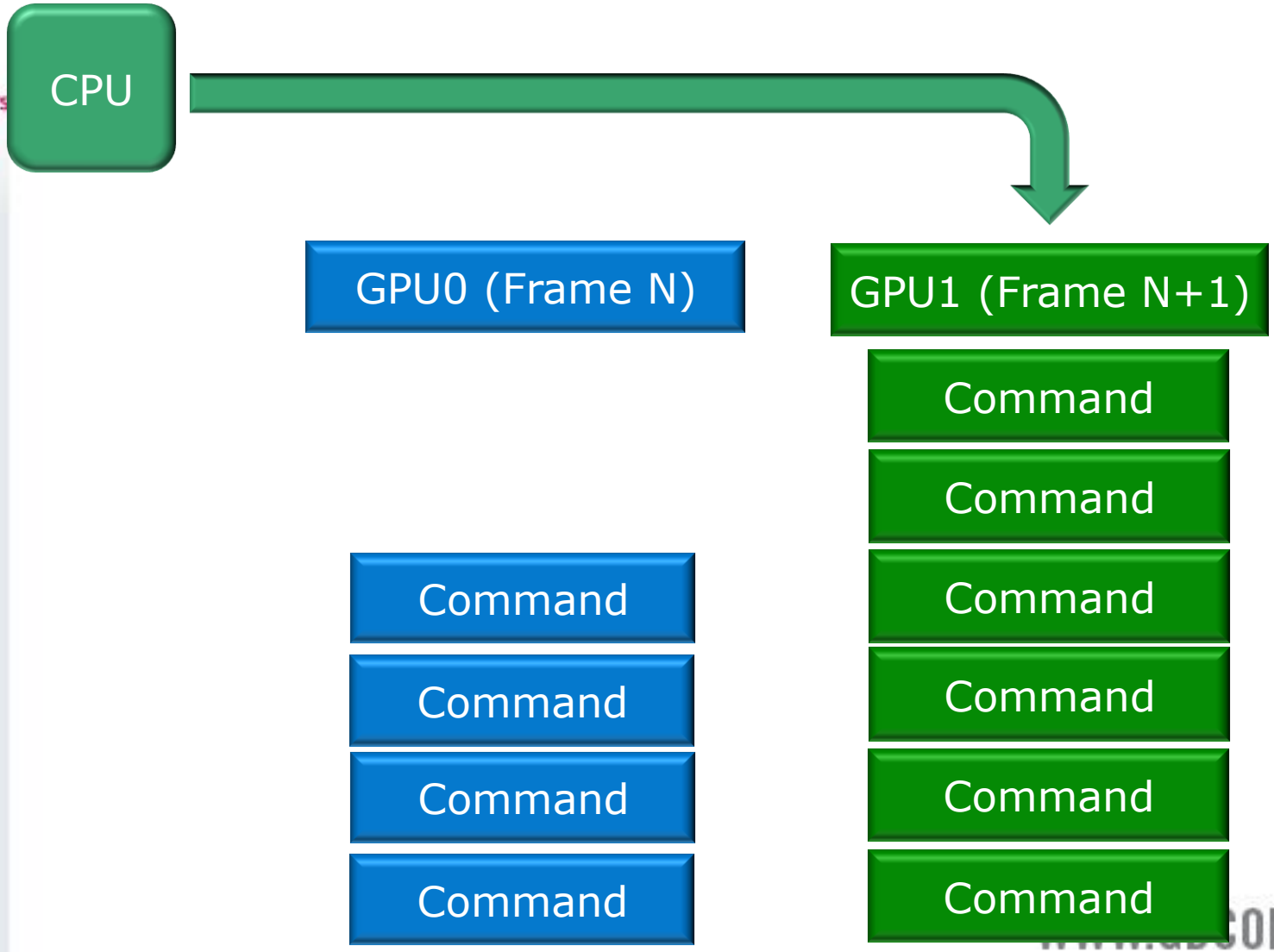
Pitfall: Waiting on Queries



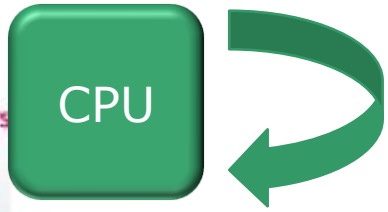
Pitfall: Waiting on Queries



Pitfall: Waiting on Queries



Pitfall: Waiting on Queries



Waiting for Query Result!!!

GPU0 (Frame N)

GPU1 (Frame N+1)

Command

Command

Command

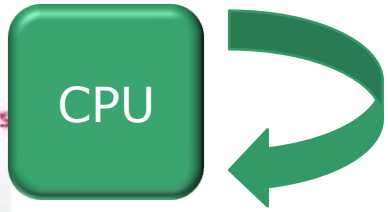
Command

Command

Command

Command

Pitfall: Waiting on Queries



Waiting for Query Result!!!

GPU0 (Frame N)

GPU1 (Frame N+1)

Command

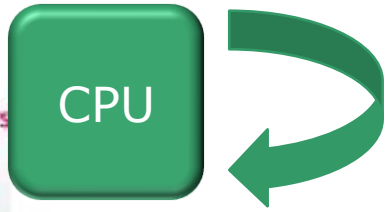
Command

Command

Command

Command

Pitfall: Waiting on Queries



Waiting for Query Result!!!

GPU0 (Frame N)

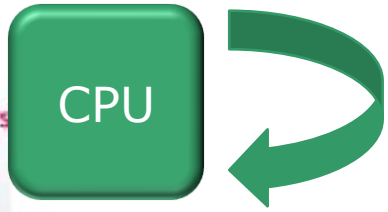
GPU1 (Frame N+1)

Command

Command

Command

Pitfall: Waiting on Queries

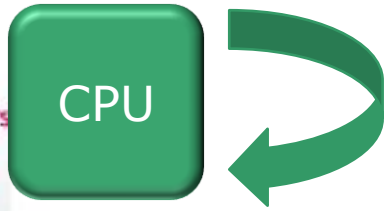


Waiting for Query Result!!!

GPU0 (Frame N)

GPU1 (Frame N+1)

Pitfall: Waiting on Queries



Waiting for Query Result!!!

GPU0 (Frame N)

GPU1 (Frame N+1)

Waiting starves GPU queues

Waiting limits parallelism

Waiting => CPU limitation

Solutions for Queries

- Avoid using queries wherever possible
- Make sure that a matched pair of BeginQuery / EndQuery calls occur within the same frame
- Avoid waiting on query results
- For queries issued every frame
  - Create additional query objects for each GPU
  - Cycle through them
- Pickup the result of a query at least N-GPU frames after it was issued
- For occlusion queries consider a CPU based approach

Pitfall: CPU Access to a Renderable Resource

- When the CPU locks a renderable resource it must wait for all GPUs to finish using the resource before acquiring the pointer
- All GPUs now have to wait until the CPU unlocks the resource pointer
- After the unlock the driver has to update the resource on each GPU via P2P copies
- Just don't do this – it destroys performance even on a single GPU setup, and is catastrophic for MGPUs

Solutions: Locks / Maps

- In DX10 stream to and copy from STAGING textures
- In DX9 StretchRect() is always better than Lock()
- At resource creation time use the appropriate flags from:
  - D3D10_USAGE
  - D3D10_CPU_ACCESS_FLAG
- AMD:In DX9 never lock static Vertex/Index Buffers because it will cause P2P copies

Concluding Pitfalls & Solutions

- Drivers take a conservative approach
  - Performing P2P copies
- You know the application best
  - Determine if a P2P copy is necessary
  - Talk to us about an app profile
- Never use resources shared between devices on DX10
  - No way to detect update by other app

Call to Action

- MGPUs provide demonstrable performance/quality gains even on low end setups
- Plan from day one to make your rendering scale
  - Use AMD library to detect number of GPUs
  - Use NVIDIA library to detect MGPU topology
- Regularly check for AFR unfriendly behavior
- Talk to us holger.gruen@amd.com
illamas@nvidia.com

Please Fill In The Feedback Forms

Questions?